

**实验报告**



**题目：词法分析程序的设计与实现**

**班 级： 2022211304**

**学 号： 2022211119**

**姓 名： 赵宇鹏**

**学 院： 计算机学院（国家示范性软件学院）**

**2024年 10 月 7 日**

## 1 实验题目及要求：

**1.1实验题目**

1．选定源语言，比如：C、Pascal、Python、Java 等，任何一种语言均可；

2．可以识别出用源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。

3．可以识别并跳过源程序中的注释。

4．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。

5．检查源程序中存在的词法错误，并报告错误所在的位置。

6．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。

1.2实现方法要求

编写LEX源程序，利用LEX编译程序生成词法分析程序。

1.3 代码运行方法

打开cmd，读取到当前lex文件所在路径，输入指令：

flex lex\_analt.l

在当前目录下生成一个名为lex.yy.c的文件

再输入指令：

gcc lex.yy.c

对该c文件进行编译，生成一个a.exe文件

输入指令：

a.exe 输入文件路径 即可成功运行程序输出结果。

## 2 程序设计说明：

2.1变量设置

1. letters:记录总字符数。
2. lines:记录总行数。
3. id\_count:记录标识符的数目。
4. int\_count:记录整数数目。
5. int\_count:记录整数数目。
6. float\_count:记录浮点数数目。
7. op\_count:记录操作符数目。
8. key\_count:记录关键字数目。
9. comment\_count:记录注释数目。
10. char\_count:记录字符常量数目。
11. string\_count:记录字符串常量数目。
12. error\_count:记录错误数目。
13. delim\_count:记录分隔符数目。
14. other\_count:记录其他符号数目。

2.2词法规则

字母（为方便判断标识符此处直接加上了\_）

letter          [a-zA-Z\_]

数字

digit           [0-9]

标识符

id              {letter}({letter}|{digit})\*

十六进制数：以0开头加一个大写或小写的x，后面接若干个0-9或a-f的数

hex\_number      0[xX][0-9a-fA-F]+

八进制数：以0开头后面接若干个0-7的数

oct\_number      0[0-7]+

整数

int\_number      {digit}+|{digit}+[eE][+-]?{digit}+

浮点数

float\_number    {digit}+[.]{digit}+|{digit}+[.]{digit}+[eE][+-]?{digit}+

其他形式数：主要为一个十进制数后面加上大小写的u,l,ll来表达无符号类型，长整型以及长长整型，或组合起来表达无符号长整型等的数。

other\_num       {digit}+[uUlL]|{digit}+ll[uU]|{digit}+LL[uU]|{digit}+[Ll][uU]|{digit}+[uU][Ll]|{digit}+[uU]ll|{digit}+[uU]LL

字符串：被“”包括的一系列符号，期间可能有转义字符。

string          \"([^\\\"]|\\.)\*\"

字符：与字符串类似，不过将“”换成‘’。

char            \'([^\\\']|\\.)\*\'

行注释：以//开头的除去\n的一系列字符

comment\_line    \/\/[^\n]\*

块注释：被/\*\*/包括的一系列字符

分隔符

delim           [(){}\[\];,?:]

占位符

space          [ \t\r]+

关键字

char\* keywords[] = {"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","typedef","union",

      "unsigned","void","volatile","while"}

操作符operator        [+\-\*/%=!><&|~\^]|"++"|"+="|"--"|"-="|"->"|"/="|"%="|"<<"|"<<="|">>"|">>="|"&&"|"&="|"|="|"^="|"||"|"<="|">="|"=="|"!="|"?:"

其他符号：不属于上述符号的符号，即无法识别的符号。

.

2.3匹配处理

1. 行注释：

根据行注释的词法规则，yytext中存有//以及后续直到换行符\n的所有字符，由于需要跳过注释，所以直接略去即可，给comment\_count加一并将letters字符总数目加上yytext的长度即可。

1. 块注释：

块注释的处理与行注释类似，对块注释中的内容直接略过一直到\*/，如果一直读取到文件结束也没有读取到\*/，那么就返回未闭合的块注释错误，并把这之间的字符全部都识别为注释略过，遇到换行符时行数加一，letters每遇到一个字符就加一即可。

1. 标识符、关键字、分界符、各类数字、占位符、操作符

识别到这些种类的记号后直接在对应的计数变量加一，总字符数加上记号相应长度，输出对应信息。

1. 字符串与字符

这二者处理类似，在对应字符串/字符数加一，总字符数加上其长度，输出对应信息，但是如果没有对应的右”/’使之闭合，则会返回错误未闭合的字符/字符串。

1. 换行符

识别到环航渡后行数加一，字符总数加二。

1. 预处理

对于预处理内容，经查询资料，我认为是预处理器应该处理的内容，所以对词法分析器来说直接跳过即可，总字符数增加，其他不变。

1. 多字节字符

由于lex识别是逐个字符进行的，所以对多字节字符会被拆分为多个符号并输出乱码，所以需要对二字节，三字节以及四字节进行特殊对待和输出，由于C语言无多字节字符的词法，所以直接报错输出即可。

8、其他字符

对于表达式没有定义的其他符号，都使用.来匹配，输出对应的识别结果以及未知字符的错误报告。

## 3 测试报告：

测试1：

**输入：**

#include<stdio.h>

int main() {

    int a,b,c;

    printf("Please input a,b,c:");

    printf("Hello world\n");

}

主要测试程序能否正常运行。

**输出：**

keyword:int

id:main

delim:(

delim:)

delim:{

keyword:int

id:a

delim:,

id:b

delim:,

id:c

delim:;

id:printf

delim:(

string:"Please input a,b,c:"

delim:)

delim:;

id:printf

delim:(

string:"Hello world\n"

delim:)

delim:;

delim:}

letters:116

lines:6

id\_count:6

int\_count:0

float\_count:0

op\_count:0

key\_count:2

comment\_count:0

char\_count:0

string\_count:2

delim\_count:13

other\_count:6

error\_count:0

**分析：**

总字符数116，与vscode统计一致，各记号类型分析正确，程序运行正常。

测试2：

**输入：**

//test 2

、、

/\*ahcnaac cjeicj\*/

0xop

/\*\*\*\*\*\*/ \*/

a+++++b

"'\n'"

{\*/   /\* asdaev\*

本输入主要测试程序错误处理能力以及识别注释的能力，第二行有两个、字符无法被识别，第四行应该识别前面的注释，后面的\*/只能被识别为操作符，最后一行的块注释没有闭合。

**输出：**

Error: Unknown multi-byte character starting with byte 0xE3 in line 2

Error: Unknown multi-byte character starting with byte 0xE3 in line 2

int:0

id:xop

op:\*

op:/

id:a

op:++

op:++

op:+

id:b

string:"'\n'"

delim:{

op:\*

op:/

Error: comment block not closed at line 8

letters:86

lines:7

id\_count:3

int\_count:1

float\_count:0

op\_count:7

key\_count:0

comment\_count:1

char\_count:0

string\_count:1

delim\_count:1

other\_count:2

error\_count:3

**分析：**

程序对多字节字符报错并输出了其字节内容，对未闭合注释也进行了报错，识别正确。

测试3：

**输入：**

include<bits/stdc++.h>

char other1[3] = {'|' };

':'

'('};

string { "auto"

kind = { {0, "KEYWORD"}

a <<= 1;

a >> b;

a >= b

a = b;

a == b;

point.first

point->first

主要用于测试一些符号及操作符的识别

**输出：**

id:include

op:<

id:bits

op:/

id:stdc

op:++

MAO:.

id:h

op:>

keyword:char

id:other1

delim:[

int:3

delim:]

op:=

delim:{

char:'|'

delim:}

delim:;

char:':'

char:'('

delim:}

delim:;

id:string

delim:{

string:"auto"

id:kind

op:=

delim:{

delim:{

int:0

delim:,

string:"KEYWORD"

delim:}

id:a

op:<<=

int:1

delim:;

id:a

op:>>

id:b

delim:;

id:a

op:>=

id:b

id:a

op:=

id:b

delim:;

id:a

op:==

id:b

delim:;

id:point

MAO:.

id:first

id:point

op:->

id:first

letters:173

lines:13

id\_count:20

int\_count:3

float\_count:0

op\_count:12

key\_count:1

comment\_count:0

char\_count:3

string\_count:2

delim\_count:16

other\_count:22

error\_count:0

测试4：

**输入：**

int main()

{

'\''

    while (inc(index, length) != -1)

    {

    char s = 'astd\n';

    char\* s = "hello world\t";

    file.close(); //

    return 0;

    "daf

}

本输入主要着重于对字符和字符串进行测试，其中还在最后有一个未闭合的字符串。

**输出：**

keyword:int

id:main

delim:(

delim:)

delim:{

char:'\''

keyword:while

delim:(

id:inc

delim:(

id:index

delim:,

id:length

delim:)

op:!=

op:-

int:1

delim:)

delim:{

keyword:char

id:s

op:=

char:'astd\n'

delim:;

keyword:char

op:\*

id:s

op:=

string:"hello world\t"

delim:;

id:file

MAO:.

id:close

delim:(

delim:)

delim:;

keyword:return

int:0

delim:;

Error: Unclosed string in line 10

delim:}

letters:170

lines:11

id\_count:8

int\_count:2

float\_count:0

op\_count:5

key\_count:5

comment\_count:1

char\_count:2

string\_count:1

delim\_count:16

other\_count:21

error\_count:1

测试5：

**输入：**

0xAAF

0XAAF

0B1101

011122378

1234

11u

11LLu

1e-10

本输入主要测试各类型数字

**输出：**

int:0xAAF

int:0XAAF

int:0

id:B1101

int:011122378

int:1234

int:11u

int:11LLu

int:1e-10

letters:56

lines:8

id\_count:1

int\_count:8

float\_count:0

op\_count:0

key\_count:0

comment\_count:0

char\_count:0

string\_count:0

delim\_count:0

other\_count:0

error\_count:0

## 4 实验总结及分析：

本次实验使用lex工具自动生成了词法分析器，完成了实验的基本要求，程序运行正常，上述5个测试也从多角度检验了程序各功能的实现，通过灵活的正则表达式定义和详细的错误处理机制，程序能够应对各种常见的词法分析任务，并生成易于理解的分析结果，高效而准确。

我也认识到lex工具相对于手工编写代码确实方便很多，工作量也小了很多，通过正则表达式来进行匹配十分简便，但是相对于手工编写灵活性略有不足，正则表达式也比较抽象难以直接写出。

在本程序，我吸取了手工编写的教训，把记号提前分好类，做好规划，补充了手工程序的不足，对lex生成程序进行了更好的构建，最终完成了本程序的撰写，对词法分析，lex的使用有了更深的理解。