

**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 5 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

采用 C/C++作为实现语言，手工编写词法分析程序。

## **2 程序设计说明：**

**2.1语言说明**

C语言拥有以下记号和单词：

1. 标识符：以字母或下划线开头，后跟字母、数字或下划线组成的符号串。
2. 关键字：具有特殊含义的保留字，本程序采用C90标准，共32个关键字。
3. 整数
4. 浮点数
5. 运算符
6. 分隔符 等等。

标识符的正规表达式为

整数的正规表达式为 digits->digit reaminder

无符号数：num->digit num1

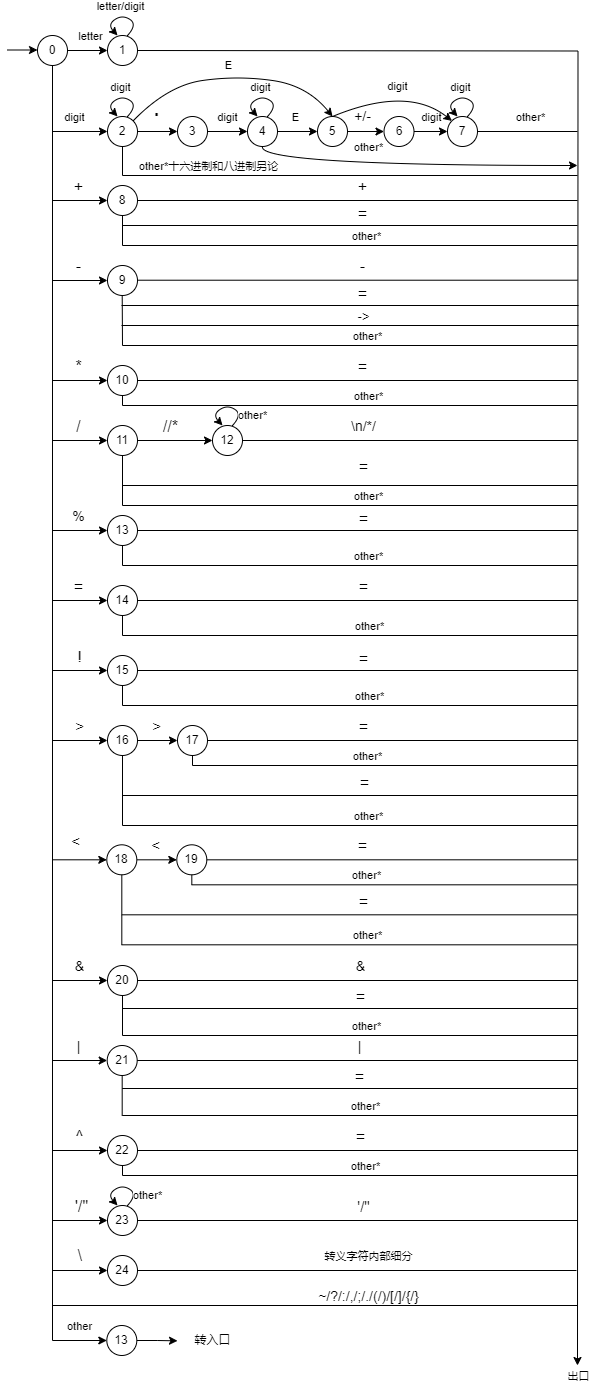
num1->digit num1 | .num2 |E num4

num2->digit num3

num3->digit num3|E num4|  
 num4->+digits|-digits|digit num5

digits->digit num5

num5->digit num5|

依此类推，其余正规表达式不再展示，将表达式转换为状态转移图即为下图所示

**2.2数据结构说明**

编写词法分析程序时，我通过前面的正规表达式和状态转移图等将记号分为以下六类：

1. id,标识符
2. key,关键字
3. int number,整数常量
4. float number,浮点数常量
5. 其他，包括字符常量，字符串常量，标点符号，转义字符以及分隔符等。
6. 错误，未识别和出现错误的记号， 本程序主要能识别出以下错误：
7. 无法识别的记号。
8. 不合法的数字，如不合法的十六进制以及八进制数。
9. 未闭合的块注释。
10. 不合法的字符常量，如未闭合以及长度大于一。
11. 未闭合的字符串。
12. 无效的\行拼接操作。

    set<string> keywords = {

        "auto", "break", "case", "char", "const",

        "continue", "default", "do", "double", "else",

        "enum", "extern", "double", "for", "goto", "if",

        "int", "long", "register", "return", "short", "signed",

        "sizeof", "static", "struct", "switch", "typedef", "union",

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

    };

所有的关键字都储存在keywords里，当进入标识符状态后，直接对keywords进行查找匹配，若匹配成功则该记号种类为关键字，否则为标识符。

    char buffer1[BUFFERSIZE];

    char buffer2[BUFFERSIZE];

int current; //缓冲区下标

  int buffer\_state = 0; //缓冲区状态

为了得到记号的确切种类和性质，只从当前字符本身不能做准确判定，需要进行超前扫描才能做出准确分析，因此需要设置缓冲区来保存输入符号串。buffer1和buffer2分别为两个输入缓冲区，大小BUFFERSIZE设为4k，用于存储输入字符，current即为向前指针所对应的缓冲区数组下标，buffer\_state则代表当前所在缓冲区号，初始为0。

int id\_num = 0;

    int int\_num = 0;

    int float\_num = 0;

    int op\_num = 0;

    int key\_num = 0;

int other\_num = 0;

我将记号分为标识符id，整数常量int，浮点数常量float，操作符op，关键字key以及其他来对字符进行计数。

    vector<string> symbol\_table;

   vector<pair<string, string>> table; //对识别出的记号以二元表的形式存储

    vector<string> errors;

我将标识符储存在symbol\_table表中，将记号的类别和属性以二元组的形式储存在table表中，错误则将错误的位置以及发生的错误信息储存在errors中。

**2.3类成员函数说明：**

以下为我使用C++语言所写LexicalAnalysis类的所有成员函数：

    void get\_char(); //每调用一次根据向前指针从输入缓冲区读取一个字符并放在c中然 后移动current

    void fill\_buffer(int buffer\_number); //填充缓冲区

    void get\_nbc(); //每次调用检查C中字符是否为空格，若是则反复调用get\_char直至 C中字符不为空格

    void cat(); //将C中字符连接到token的字符串后

    void retract(); //向前指针回退一个字符

    bool is\_letter(); //判断C中字符是否为字母

    bool is\_digit(); //判断C中字符是否为数字

    bool is\_reserved(string token); //判断token是否为保留字

    string\* symbol\_insert(string token); //将token插入标识符表

    void table\_insert(string mark, string attr); //将token插入二元表

    void error(string problem); //报告错误

    void output(); //输出结果

    void run();

    LexicalAnalysis(string filename) {

        for(int i = 0; i < BUFFERSIZE; i++) {

            buffer1[i] = -1;

            buffer2[i] = -1;

        }

        buffer1[BUFFERSIZE] = '\0';

        buffer2[BUFFERSIZE] = '\0';

        file.open(filename, ios::in);

        if (!file.is\_open()) {

            cout << "Error opening file";

            exit(1);

        }

};

###### 2.3.1 get\_char

get\_char函数的作用是每调用一次该函数都会在向前指针所指缓冲区位置读入一个字符进入词法分析程序，然后移动向前指针，因此在初始状态时需要首先读取文件填充缓冲区，在向前指针指到缓冲区末尾时再填充另一个缓冲区，以此进行循环填充使用缓冲区，读取字符的同时还可以直接统计所有字符数目，对\t制表符字符总数加四，对于windows系统下的换行符，其占两个字符，所以为方便统计验算，本程序内的换行符也占两个字符，总字符数加二，对于汉字，由于其在UTF-8中占一到三字节不等，所以get\_char函数还使用与操作对当前字符是否为多字节字符进行判断，以方便字符总数的计数，具体实现请看源代码附件。

###### **2.3.2 fill\_buffer**

该函数的目的为填充缓冲区，当get\_char函数判断读取到了缓冲区末尾就调用此函数并对另一缓冲区进行填充，然后在数据末尾加上\0终结符。

###### **2.3.4 get\_nbc**

每次调用检查C中字符是否为空格，若是则反复调用get\_char直至C中字符不为空格,以免在词法分析主程序中进行匹配降低效率。

###### **2.3.5 cat**

将C中字符连接到token中，方便符号表和标识符表的插入。

###### **2.3.6 retract**

由于很多记号的分析需要进行单个字符的超前扫描，所以当超前扫描结束未匹配到预期字符后需要调用retract函数将向前指针回退一步来进行下一步的词法分析。

###### **2.3.7 is\_letter**

判断当前字符是否为字母。

###### **2.3.8 is\_digit**

判断当前字符是否为数字。

###### **2.3.9 is\_reserved**

判断输入的token是否为关键字。

###### **2.3.10 symbol\_insert**

对已经判断为标识符的记号，将其插入进标识符表symbol中并返回其在标识符表中所在位置。

###### **2.3.11 table\_insert**

将分析好的记号类别及其属性储存在记号表中。

###### **2.3.12 error**

将发现的错误所在位置以及错误信息储存到errors中以便后续输出。

###### **2.3.13 output**

输出统计的总行数，总字符数，总标识符数，总整数常量数，总浮点数常量数，总操作符数，总关键字数以及其他字符数，并输出标识符表，记号表以及所有的错误。

**2.4 词法分析器**

初始状态为state0，通过get\_char读取下一字符后与预设状态进行匹配并进行分析，当读取到文件末尾EOF时退出循环执行输出操作。

###### **2.4.1 操作符，标点和分隔符**

在遇到操作符，标点和分隔符时，词法分析器通过向前查看缓冲区中的字符来判断符号的类型，注意匹配是最大长度匹配以及优先匹配，即优先匹配最长的符合的记号，若长度相等则匹配排前面的符号，如表达式a+++++b会被识别为a, ++, ++, +, b，某些记号没有与其他记号相同的前缀字符则可识别到就直接插入记号表中无需超前扫描，如(, ), [, ]等。

###### **2.4.2 标识符或关键字**

当读取到字母或下划线时，程序进入识别标识符或关键字状态，并对下一字符进行扫描，若为字母，数字或下划线则将其加入token中并继续往下扫描，否则向前指针回退，在关键字中查找token，若查找成功则为关键字，否则为标识符，并将相关信息储存进相应表中。

###### **2.4.3 数字常量**

当读取到数字时，进入数字识别状态，由于C语言支持八进制，十进制以及十六进制数字，所以需要检查token的前缀，若为0x则作为十六进制数处理，若为0o则作为八进制处理，否则默认按照十进制处理，对于十六进制和八进制，程序只考虑了其整数数字，若有不合法字符则将相关信息插入errors中，对于十进制则需要一直往后读取至其他字符，则将其分为整数类型并存储，若读取到e/E,则进入读取指数状态，若读取到.则进入到识别浮点数状态，如果有e/E,其后跟随一个可选的+或-，之后跟随至少一个十进制数字。如果此时读取到缓冲区结束，则返回十进制浮点数记号。否则判断剩余的部分是否是合法的浮点数后缀，如果是，则返回十进制浮点数记号，否则返回错误。

###### **2.4.4 字符常量**

当读取到符号‘时，进入字符分析状态，此后，词法分析器不断读取字符，

如果遇到'则结束，如果遇到换行符或者文件结尾，则返回未闭合字符常量错误。

如果遇到了转义字符则进入转义字符状态，并将char\_flag置一表示之后还要返回字符处理状态，其余字符则结束分析并储存字符常量。

###### **2.4.5 字符串**

字符串与字符处理同理，将’换成”即可。

###### **2.4.6行注释与块注释**

当匹配到/时进入注释处理模块，进行超前扫描，若为\*则进入块注释处理，此时程序会不断往后读取字符直到遇到\*/，若读取到文件结束则返回注释未闭合错误；若为/则进入行注释处理，一直读取字符到换行或文件末尾即可，对于注释，为方便比对，本程序统计了注释的字符数，对英文字符以及汉字都能正确统计数目。

###### **2.4.7转义字符\**

\具有行拼接和转义两个作用，对行拼接若往后读取字符是换行符则直接忽视换行符转回state = 0初始状态继续读取，否则返回未闭合的转义字符错误，对转义作用，则对相应字符进行匹配，若匹配失败则返回无效的转义字符错误，成功则存入table表中。

## **3 运行结果与分析说明**

###### **测试一：**

本测试用例用于测试程序能否正常运行

**输入：**

#include<stdio.h>

int main()

{

    int a,b;

    scanf("%d%d",&a,&b);

printf("%d\n",a+b);

printf("\"hello world!\"");

    return 0;

}

**输出：**

Total lines: 9

Total letters: 148

Total identifiers: 10

Total int numbers: 1

Total float numbers: 0

Total operators: 3

Total keywords: 3

Total others: 25

Symbol table:

main a b scanf printf

Table:

<KEY , int>

<ID , main>

<( , >

<) , >

<{ , >

<KEY , int>

<ID , a>

<, , >

<ID , b>

<; , >

<ID , scanf>

<( , >

<STRING , "%d%d">

<, , >

<BOP , &>

<ID , a>

<, , >

<BOP , &>

<ID , b>

<) , >

<; , >

<ID , printf>

<( , >

<ESCAPE , \n>

<STRING , "%d\n">

<, , >

<ID , a>

<AOP , +>

<ID , b>

<) , >

<; , >

<ID , printf>

<( , >

<ESCAPE , \">

<ESCAPE , \">

<STRING , "\"hello world!\"">

<) , >

<; , >

<KEY , return>

<NUM , 0>

<; , >

<} , >

Errors:

**分析：**程序共9行，含有关键字int和return共三个，标识符main, scanf, a, b, printf共10个，字符数148与vscode所统计一致，整数0一个，能正确读取字符串内容以及标点符号和分界符，字符串中的转义字符也能够正确识别，运行正常。

###### **测试二：**

在测试一输入的基础上进行适当的改错，检测排错能力

**输入：**

#include<stdio.h>

int main()

{

    int a,b;

    float c = 1.b; //错误的浮点数

    int d = 0xo2; //具有非法符号的十六进制数

    int e = 0xa2; //正确的十六进制数形成对照

    float f = 1.2e3; //正确的浮点数

    scanf("%d%d",&a,&b);

    printf("%d\n\8",a+b); //\8为\8为非法转义字符

    printf("\"hello world!\"); //未闭合的”

    return 0;

}

**输出：**

Total lines: 14

Total letters: 236

Total identifiers: 15

Total int numbers: 3

Total float numbers: 1

Total operators: 7

Total keywords: 6

Total others: 26

Symbol table:

main a b float c d e f scanf printf

Table:

<KEY , int>

<ID , main>

<( , >

<) , >

<{ , >

<KEY , int>

<ID , a>

<, , >

<ID , b>

<; , >

<ID , float>

<ID , c>

<AOP , assign-op>

<; , >

<KEY , int>

<ID , d>

<AOP , assign-op>

<NUM , 2>

<; , >

<KEY , int>

<ID , e>

<AOP , assign-op>

<NUM , 0xa2>

<; , >

<KEY , float>

<ID , f>

<AOP , assign-op>

<NUM , 1.2e3>

<; , >

<ID , scanf>

<( , >

<STRING , "%d%d">

<, , >

<BOP , &>

<ID , a>

<, , >

<BOP , &>

<ID , b>

<) , >

<; , >

<ID , printf>

<( , >

<ESCAPE , \n>

<STRING , "%d\n\8">

<, , >

<ID , a>

<AOP , +>

<ID , b>

<) , >

<; , >

<ID , printf>

<( , >

<ESCAPE , \">

<ESCAPE , \">

<KEY , return>

<NUM , 0>

<; , >

<} , >

Errors:

Error at line 5 position 17: Invalid float number

Error at line 6 position 15: Invalid hex number

Error at line 10 position 18: Invalid escape character

Error at line 11 position 31: Invalid string or char

**分析：**

通过输出可以看到程序正确识别了新增的十六进制数以及浮点数，通过一次扫描，对错误的十六进制数以及浮点数，无法识别的转义字符，未闭合的字符串双引号错误也进行了报错提示并输出了错误位置和相关信息。

测试三：

**输入：**

//测试三

struct Node

{

    char name[20];

    int id;

} Node;

int main() {

    struct Node node;

    node.id = 123;

    (&node)->id = 456;

    int a = '2';

    int num\_1,\_num2;

    float b = 5.3e-2;

    printf("%d +%f = %f",a,b,a+b);

    /\*

    return 0;

}

**输出：**

Total lines: 19

Total letters: 273

Total identifiers: 20

Total int numbers: 3

Total float numbers: 1

Total operators: 8

Total keywords: 8

Total others: 27

Symbol table:

Node name id main node a num\_1 \_num2 b printf

Table:

<KEY , struct>

<ID , Node>

<{ , >

<KEY , char>

<ID , name>

<[ , >

<NUM , 20>

<] , >

<; , >

<KEY , int>

<ID , id>

<; , >

<} , >

<ID , Node>

<; , >

<KEY , int>

<ID , main>

<( , >

<) , >

<{ , >

<KEY , struct>

<ID , Node>

<ID , node>

<; , >

<ID , node>

<MAO , .>

<ID , id>

<AOP , assign-op>

<NUM , 123>

<; , >

<( , >

<BOP , &>

<ID , node>

<) , >

<AROP , ->>

<ID , id>

<AOP , assign-op>

<NUM , 456>

<; , >

<KEY , int>

<ID , a>

<AOP , assign-op>

<CHAR , '2'>

<; , >

<KEY , int>

<ID , num\_1>

<, , >

<ID , \_num2>

<; , >

<KEY , float>

<ID , b>

<AOP , assign-op>

<NUM , 5.3e-2>

<; , >

<ID , printf>

<( , >

<STRING , "%d +%f = %f">

<, , >

<ID , a>

<, , >

<ID , b>

<, , >

<ID , a>

<AOP , +>

<ID , b>

<) , >

<; , >

Errors:

Error at line 19 position 0: Unclosed comment

**分析：**

本次输入对更多的关键字以及符号如.以及->进行了测试了，程序运行正常，同时程序也跳过了输入开头的行注释以及结尾的块注释，同时对没有闭合的块注释进行了报错。

## **4 总结**

本次实验中我上手编写了一个词法分析程序，使我对词法分析的流程更加清楚，对相关知识点的掌握更加牢固。为了实现C语言的词法分析，需要有标准的C语言词法规则进行参照，而我初期时只是凭借自己对C语言的印象来实现C语言的词法分析，因此遗漏了很多如十六进制，八进制数，转义字符等细节，在多次查阅资料后对这些细节进行了一一修补，并通过设计的测试样例来解决了很多bug。本次实验代码的主要框架还是按照课本上面的来搭建实现的，代码直观但是比较冗杂，仍然有很大的改进空间，希望今后的实验能够规划好代码整体结构以及相关细节，不断进步，本次实验除了让我对编译原理课程内容有了更多了解，也让我的C++能力得到了提高，受益匪浅。