实验一 词法分析程序设计与实现

**一、实验目的**

通过编写和调试一个词法分析程序，掌握在对程序设计语言的源程序进

行扫描的过程中，将字符流形式的源程序转化为一个由各类单词构成的序列

的词法分析方法。

**二、基本实验内容与要求**

假定一种高级程序设计语言中的单词主要包括关键字 begin、end、if、then、

else、while、do；标识符；浮点常数；六种关系运算符；一个赋值符和四个

算术运算符，试构造能识别这些单词的词法分析程序（各类单词的分类码可

参见表 1）。

**输入：**由符合和不符合所规定的单词类别结构的各类单词组成的源程序

文件。

**输出：**把所识别出的每一单词均按形如（CLASS，VALUE）的二元式形

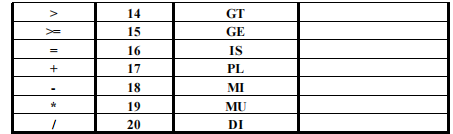
式输出，并将结果放到某个文件中。对于标识符和浮点常数，CLASS 字段为

相应的类别码的助记符；VALUE 字段则是该标识符、常数的具体值；对于关

键字和运算符，采用一词一类的编码形式，仅需在二元式的 CLASS 字段上

放置相应单词的类别码的助记符，VALUE 字段则为“空”。





**要求：**

1、上机前完成词法分析程序的程序流程设计，并选择好相应的数据结构。

2、用于测试扫描器的实例源文件中至少应包含两行以上的源代码。

3、对于输入的测试用例的源程序文件，词法正确的单词分析结果在输出

文件中以二元式形式输出，错误的字符串给出错误提示信息。

例如，若输入文件中的内容为：“if myid>=1.5 then x=y”，则输出文件

中的内容应为：

（IF， ）

（ID，’myid’）

（GE， ）

（UCON，1.5）

（THEN， ）

（ID，’x’）

（IS， ）

（ID，’y’）

**三、参考实现方法**

**1、一般实现方法说明**

词法分析是编译程序的第一个处理阶段，可以通过两种途径来构造词法

分析程序。其一是根据对语言中各类单词的某种描述或定义（如 BNF），用手

工的方式（例如可用 C 语言）构造词法分析程序。一般地，可以根据文法或

状态转换图构造相应的状态矩阵，该状态矩阵连同控制程序一起便组成了编

译器的词法分析程序；也可以根据文法或状态转换图直接编写词法分析程序。

构造词法分析程序的另外一种途径是所谓的词法分析程序的自动生成，即首

先用正规式对语言中的各类单词符号进行词型描述，并分别指出在识别单词时，词法分析程序所应进行的语义处理工作，然后由一个所谓词法分析程序

的构造程序对上述信息进行加工。如美国 BELL 实验室研制的 LEX 就是一个被

广泛使用的词法分析程序的自动生成工具。总的来说，开发一种新语言时，

由于它的单词符号在不停地修改，采用 LEX 等工具生成的词法分析程序比较

易于修改和维护。一旦一种语言确定了，则采用手工编写词法分析程序效率

更高。本实验建议使用手工编写的方法。

在一个程序设计语言中，一般都含有若干类单词符号，为此可首先为每

类单词建立一张状态转换图，然后将这些状态转换图合并成一张统一的状态

图，即得到了一个有限自动机，再进行必要的确定化和状态数最小化处理，

最后添加当进行状态转移时所需执行的语义动作，就可以据此构造词法分析

程序了。

**2、单词分类与词法分析器的设计**

为了使词法分析程序结构比较清晰，且尽量避免某些枝节问题的纠缠，

我们假定要编译的语言中，全部关键字都是保留字，程序员不得将它们作为

源程序中的标识符；在源程序的输入文本中，关键字、标识符、浮点常数之

间，若未出现关系和算术运算符以及赋值符，则至少须用一个空白字符加以

分隔。作了这些限制以后，就可以把关键字和标识符的识别统一进行处理。

即每当开始识别一个单词时，若扫视到的第一个字符为字母，则把后续输入

的字母或数字字符依次进行拼接，直至扫视到非字母、数字字符为止，以期

获得一个尽可能长的字母数字字符串，然后以此字符串查所谓保留字表（此

保留字表要事先造好），若查到此字符串，则取出相应的类别码；反之，则表

明该字符串应为一标识符。

采用上述策略后，针对表 1 中的部分单词可以参考图 1 和程序 1，用 C

语言编写出符合以上几项要求的一个扫描器程序。

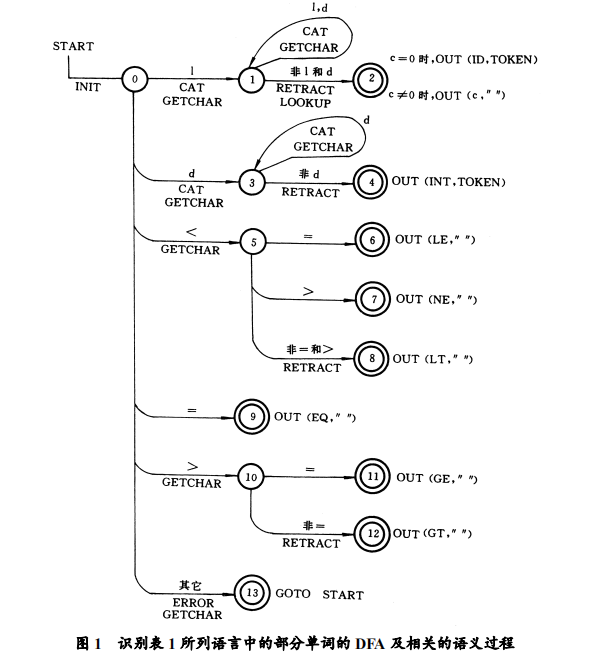


图 1 中所出现的语义变量及语义函数的含义和功能说明如下：

**函数 GETCHAR**：每调用一次，就把扫描指示器当前所指示的源程序字符

送入字符变量 ch，然后把扫描指示器前推一个字符位置。

**字符数组 TOKEN**：用来依次存放一个单词词文中的各个字符。

**函数 CAT**：每调用一次，就把当前 ch 中的字符拼接于 TOKEN 中所存字符

串的右边。

**函数 LOOKUP**：每调用一次，就以 TOKEN 中的字符串查保留字表，若查到，就将相应关键字的类别码赋给整型变量 c；否则将 c 置为零。

**函数 RETRACT**：每调用一次，就把扫描指示器回退一个字符位置（即退

回多读的那个字符）。

**函数 OUT**：一般仅在进入终态时调用此函数，调用的形式为 OUT(c，VAL)。

其中，实参 c 为相应单词的类别码助记符；实参 VAL 为 TOKEN（即词文）或

为空串。函数 OUT 的功能是，在送出一个单词的内部表示之后，返回到调用

该词法分析程序的那个程序。

**3、词法分析程序的实现**

**程序 1 根据图 1 编写的扫描器**

# include <stdio.h>

# include <ctype.h>

# include <string.h>

# define ID 6

# define INT 7

# define LT 8

# define LE 9

# define EQ 10

# define NE 11

# define GT 12

# define GE 13

char TOKEN[20]；

extern int lookup (char\*);

extern void out (int, char\*);

extern report\_error (void);

void scanner\_example (FILE \*fp)

{

char ch; int i, c;

ch=fgetc (fp);

if (isalpha (ch)) /\*it must be a identifer!\*/

{

TOKEN[0]=ch; ch=fgetc (fp); i=1;

while (isalnum (ch))

{

TOKEN[i]=ch; i++；

ch=fgetc (fp);

}

TOKEN[i]= ′＼0′

fseek(fp,-1,1); /\* retract\*/

c=lookup (TOKEN);

if (c==0) out (ID,TOKEN); else out (c," ")；

}

else

if(isdigit(ch))

{

TOKEN[0]=ch; ch=fgetc(fp); i=1;

while(isdigit(ch))

{TOKEN[i]=ch; i++;

ch=fgetc(fp);

}

TOKEN[i]= ′＼0′；

fseek(fp,-1,1);

out(INT,TOKEN);

}

else

switch(ch)

{

case ′＜′: ch=fgetc(fp);

if(ch==′=′)out(LE," ");

else if(ch==′＞′) out (NE," ");

else

{

fseek (fp,-1,1);

out (LT," ");

}

break;

case ′=′: out(EQ, " "); break;

case ′＞′: ch=fgetc(fp);

if(ch==′=′)out(GE," ");

else

{

fseek(fp,-1,1);

out(GT," ");

}

break;

default: report\_error( ); break;

}

return;

}

程序 1 中所用的若干函数以及主程序有待于具体编写，并需事先建立好

保留字表，以备查询。例如：

/\* 建立保留字表 \*/

#define MAX\_KEY\_NUMBER 20 /\*关键字的数量\*/

#define KEY\_WORD\_END “waiting for your expanding” /\*关键字结束标记\*/

char \*KeyWordTable[MAX\_KEY\_NUMBER]={“begin”,“end”, “if”, “then”, “else”, KEY\_WORD\_END};

/\* 查保留字表，判断是否为关键字 \*/

int lookup (char \*token)

{

int n=0;

while (strcmp(KeyWordTable[n], KEY\_WORD\_END)) /\*strcmp 比较两串是否相同，若相同返回 0\*/

{

if (!strcmp(KeyWordTable[n], token)) /\*比较 token 所指向的关键字和保留字表中哪个关键字相符\*/

{

return n+1; /\*根据单词分类码表 I，设置正确的关键字类别码，并返回此类别码的值\*/

break;

}

n++;

}return 0; /\*单词不是关键字，而是标识符\*/

}

**四、扩展实验**

1、扩充关键字的数目、增加逻辑运算符等单词类别、将常数再细分成字符串

常量、整型常量和无符号数常量等；添加词法分析中单词出错的位置和错

误类型，以及删除注释部分等。

2、对基本实验内容进行扩充，增加状态转换图显示、词法分析过程的显示等

可视化展现功能。

3、研读 GCC，CLANG 等开源编译器的词法分析部分，分析其程序结构、实现

方法、识别的单词分类等。

4、其它自选题目。注意自选扩展实验题目须经过实验指导教师同意并备案。

**1、基础部分：**

源代码：

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define ID 6

#define INT 7

#define LT 8

#define LE 9

#define EQ 10

#define NE 11

#define GT 12

#define GE 13

#define FLOAT 14

#define MAX\_KEY\_NUMBER 20 // 关键字的数量

#define KEY\_WORD\_END NULL // 使用NULL表示关键字结束

char TOKEN[20];

char \*KeyWordTable[MAX\_KEY\_NUMBER] = {"begin", "end", "if", "then", "else", NULL};

// 函数声明

extern int lookup(char\*);

extern void out(FILE\*, int, char\*);

extern void report\_error(FILE\*);

// 查找关键字的函数

int lookup(char \*token) {

    int n = 0;

    while (KeyWordTable[n] != KEY\_WORD\_END) {

        if (!strcmp(KeyWordTable[n], token)) {

            return n + 1; // 返回关键字类别码

        }

        n++;

    }

    return 0; // 返回0表示不是关键字

}

// 获取下一个字符的函数

char getchar\_from\_file(FILE \*fp) {

    return fgetc(fp);

}

// 将字符拼接到TOKEN中

void cat\_to\_token(char ch, int \*i) {

    TOKEN[\*i] = ch;

    (\*i)++;

    TOKEN[\*i] = '\0'; // 确保字符串以'\0'结束

}

// 词法分析器的实现

void scanner\_example(FILE \*input\_fp, FILE \*output\_fp) {

    char ch;

    int i, c;

    while ((ch = getchar\_from\_file(input\_fp)) != EOF) {

        memset(TOKEN, 0, sizeof(TOKEN)); // 清空 TOKEN

        if (isalpha(ch)) { // 标识符或关键字

            TOKEN[0] = ch;

            i = 1;

            while (isalnum((ch = getchar\_from\_file(input\_fp)))) {

                cat\_to\_token(ch, &i);

            }

            fseek(input\_fp, -1, SEEK\_CUR); // 退回一个字符

            c = lookup(TOKEN);

            if (c == 0) out(output\_fp, ID, TOKEN);

            else out(output\_fp, c, TOKEN);  // 改为输出 TOKEN 而不是空字符串

        } else if (isdigit(ch)) { // 整数或浮点数

            TOKEN[0] = ch;

            i = 1;

            int isFloat = 0; // 标记是否为浮点数

            while (isdigit((ch = getchar\_from\_file(input\_fp))) || (ch == '.' && !isFloat)) {

                if (ch == '.') {

                    isFloat = 1; // 发现小数点

                }

                cat\_to\_token(ch, &i);

            }

            fseek(input\_fp, -1, SEEK\_CUR); // 退回一个字符

            if (isFloat) {

                out(output\_fp, FLOAT, TOKEN); // 输出浮点数

            } else {

                out(output\_fp, INT, TOKEN); // 输出整数

            }

        } else {

            switch (ch) {

                case '<':

                    ch = getchar\_from\_file(input\_fp);

                    if (ch == '=') out(output\_fp, LE, " ");

                    else if (ch == '>') out(output\_fp, NE, " ");

                    else {

                        fseek(input\_fp, -1, SEEK\_CUR); // 退回一个字符

                        out(output\_fp, LT, " ");

                    }

                    break;

                case '=': out(output\_fp, EQ, " "); break;

                case '>':

                    ch = getchar\_from\_file(input\_fp);

                    if (ch == '=') out(output\_fp, GE, " ");

                    else {

                        fseek(input\_fp, -1, SEEK\_CUR); // 退回一个字符

                        out(output\_fp, GT, " ");

                    }

                    break;

                case ' ': // 忽略空格

                case '\t':

                case '\n':

                    break;

                default:

                    report\_error(output\_fp); // 报告错误

                    break;

            }

        }

    }

}

// 修改后的输出函数：将输出保存到文件

void out(FILE \*output\_fp, int code, char \*val) {

    if (code == ID) {

        fprintf(output\_fp, "(ID, '%s')\n", val);

    } else if (code == INT) {

        fprintf(output\_fp, "(INT, '%s')\n", val);

    } else if (code == FLOAT) {

        fprintf(output\_fp, "(FLOAT, '%s')\n", val);

    } else if (code >= 1 && code <= MAX\_KEY\_NUMBER) { // 关键字输出

        fprintf(output\_fp, "(KEYWORD, '%s')\n", val);

    } else {

        fprintf(output\_fp, "(%d, )\n", code);

    }

}

// 修改后的错误报告函数：将错误信息写入文件

void report\_error(FILE \*output\_fp) {

    fprintf(output\_fp, "Error: Invalid character\n");

}

int main() {

    FILE \*input\_fp = fopen("C:\\Users\\Ava\\CLionProjects\\BY\_1\\test.txt", "r"); // 输入文件

    if (input\_fp == NULL) {

        perror("Unable to open input file!");

        return 1;

    }

    FILE \*output\_fp = fopen("C:\\Users\\Ava\\CLionProjects\\BY\_1\\output.txt", "w"); // 输出文件

    if (output\_fp == NULL) {

        perror("Unable to open output file!");

        fclose(input\_fp);

        return 1;

    }

    scanner\_example(input\_fp, output\_fp);

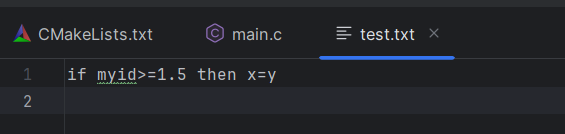
    fclose(input\_fp);

    fclose(output\_fp);

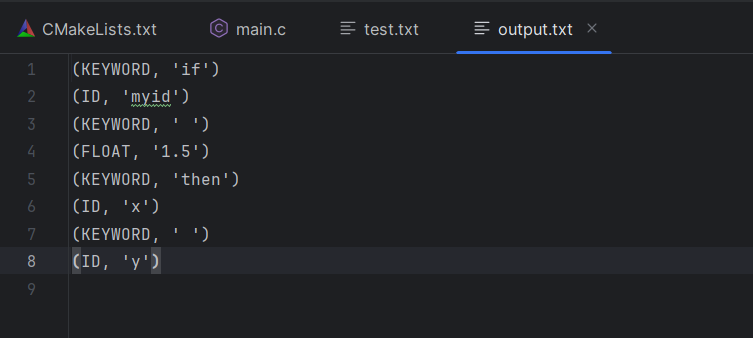
    return 0;

}

输入存于绝对路径为C:\Users\Ava\CLionProjects\BY\_1\test.txt的文本中：



输出存于绝对路径为C:\Users\Ava\CLionProjects\BY\_1\output.txt的文本中：



之后我尝试用flex再对基础实验做词法分析，教程是在csdn上找到：

<https://blog.csdn.net/weixin_44007632/article/details/108666375?fromshare=blogdetail&sharetype=blogdetail&sharerId=108666375&sharerefer=PC&sharesource=qq_74321854&sharefrom=from_link>

%{

    #include <stdio.h>

    #include <stdlib.h>

    #include <string.h>

    // 定义符号类型

    #define ID 6

    #define INT 7

    #define LT 8

    #define LE 9

    #define EQ 10

    #define NE 11

    #define GT 12

    #define GE 13

    #define FLOAT 14

    FILE \*output\_fp; // 输出文件指针

    // 查找关键字函数

    int lookup(char \*token) {

        if (strcmp(token, "if") == 0) return 1;

        if (strcmp(token, "then") == 0) return 2;

        if (strcmp(token, "else") == 0) return 3;

        if (strcmp(token, "begin") == 0) return 4;

        if (strcmp(token, "end") == 0) return 5;

        return 0;

    }

    // 输出函数

    void out(int code, char \*val) {

        if (code == ID) {

            fprintf(output\_fp, "(ID, '%s')\n", val);

        } else if (code == INT) {

            fprintf(output\_fp, "(INT, '%s')\n", val);

        } else if (code == FLOAT) {

            fprintf(output\_fp, "(FLOAT, '%s')\n", val);

        } else if (code >= 1 && code <= 5) {

            fprintf(output\_fp, "(KEYWORD, '%s')\n", val);

        } else {

            fprintf(output\_fp, "(%d, )\n", code);

        }

    }

    %}

    %%

    [ \t\n]+             ;    // 忽略空格和换行符

    "if"|"then"|"else"|"begin"|"end" {

                            out(lookup(yytext), yytext);

                        }

    [0-9]+\.[0-9]+       { out(FLOAT, yytext); }  // 匹配浮点数

    [0-9]+               { out(INT, yytext); }    // 匹配整数

    [a-zA-Z\_][a-zA-Z0-9\_]\* {

                            int code = lookup(yytext);

                            if (code == 0) out(ID, yytext); // 非关键字的标识符

                            else out(code, yytext);         // 关键字

                        }

    "<="                { out(LE, " "); }         // 匹配 <=

    ">="                { out(GE, " "); }         // 匹配 >=

    "<"                 { out(LT, " "); }         // 匹配 <

    ">"                 { out(GT, " "); }         // 匹配 >

    "="                 { out(EQ, " "); }         // 匹配 =

    .                   { fprintf(output\_fp, "Error: Invalid character '%s'\n", yytext); } // 错误字符处理

    %%

    // 定义 yywrap 函数

    int yywrap() {

        return 1;

    }

    int main() {

        FILE \*input\_fp;

        // 打开输入文件，指定路径 D:\Flex\input.txt

        input\_fp = fopen("D:\\Flex\\input.txt", "r");

        if (input\_fp == NULL) {

            perror("Unable to open input file!");

            return 1;

        }

        yyin = input\_fp;

        // 打开输出文件，指定路径 D:\Flex\output.txt

        output\_fp = fopen("D:\\Flex\\output.txt", "w");

        if (output\_fp == NULL) {

            perror("Unable to open output file!");

            fclose(input\_fp);

            return 1;

        }

        // 执行词法分析

        yylex();

        // 关闭文件

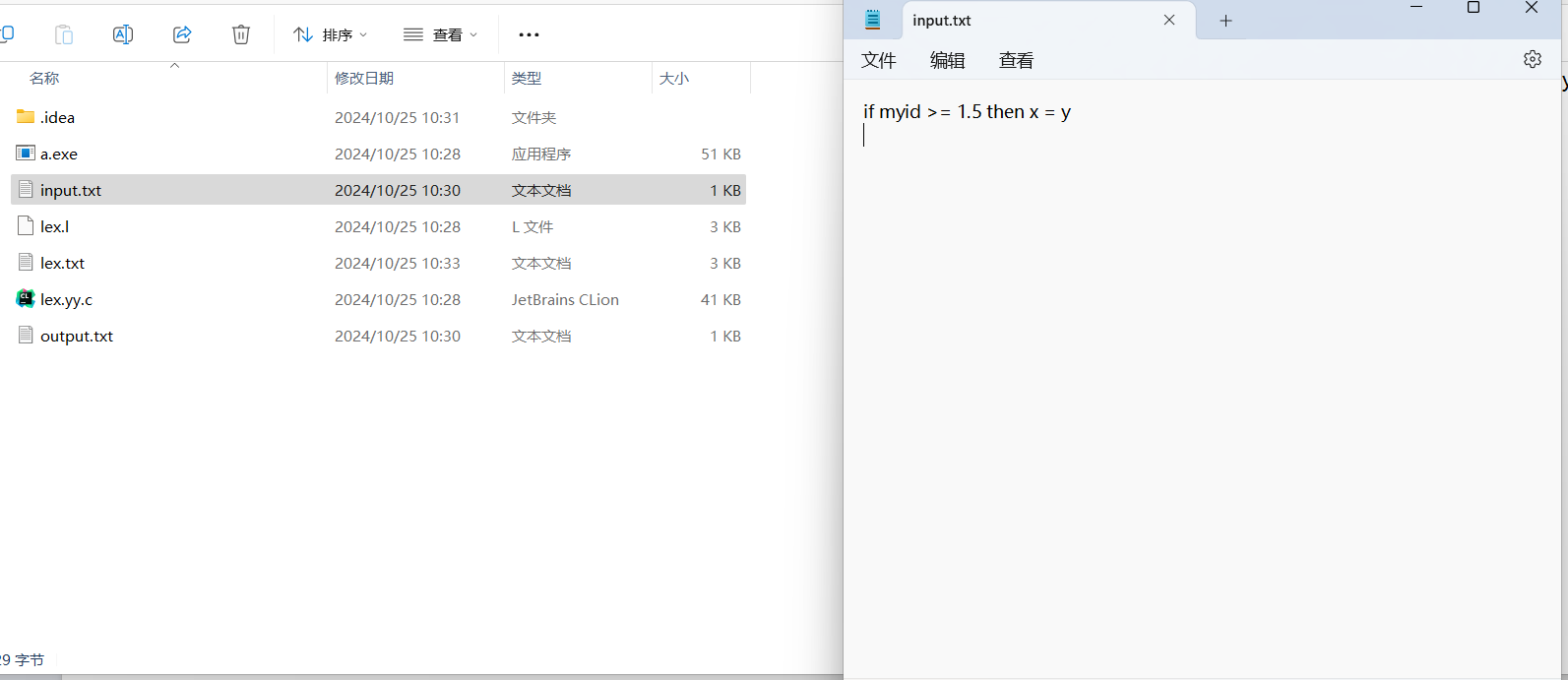
        fclose(input\_fp);

        fclose(output\_fp);

        return 0;

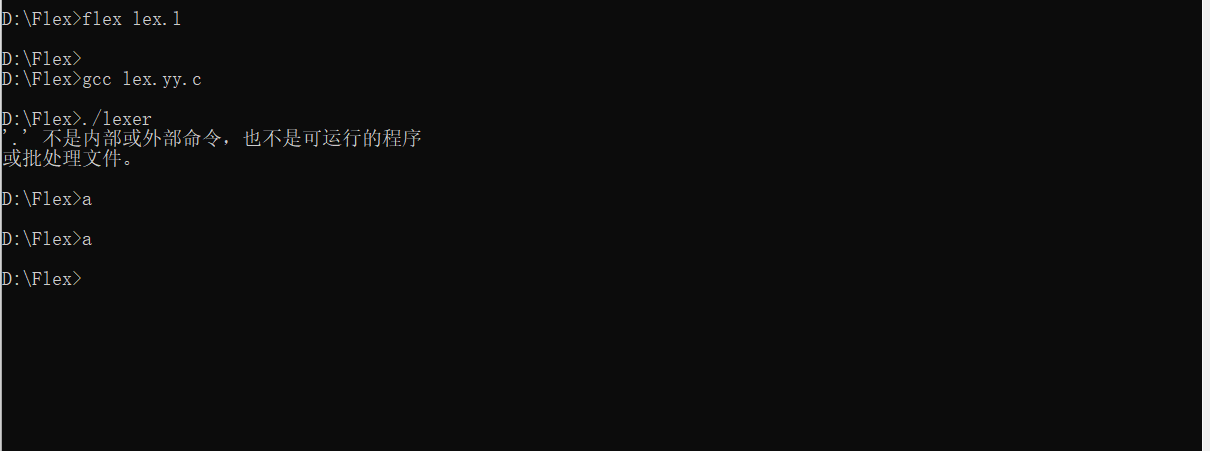
    }

将文本文件改成“lex.l”，并提前创建好input.txt：

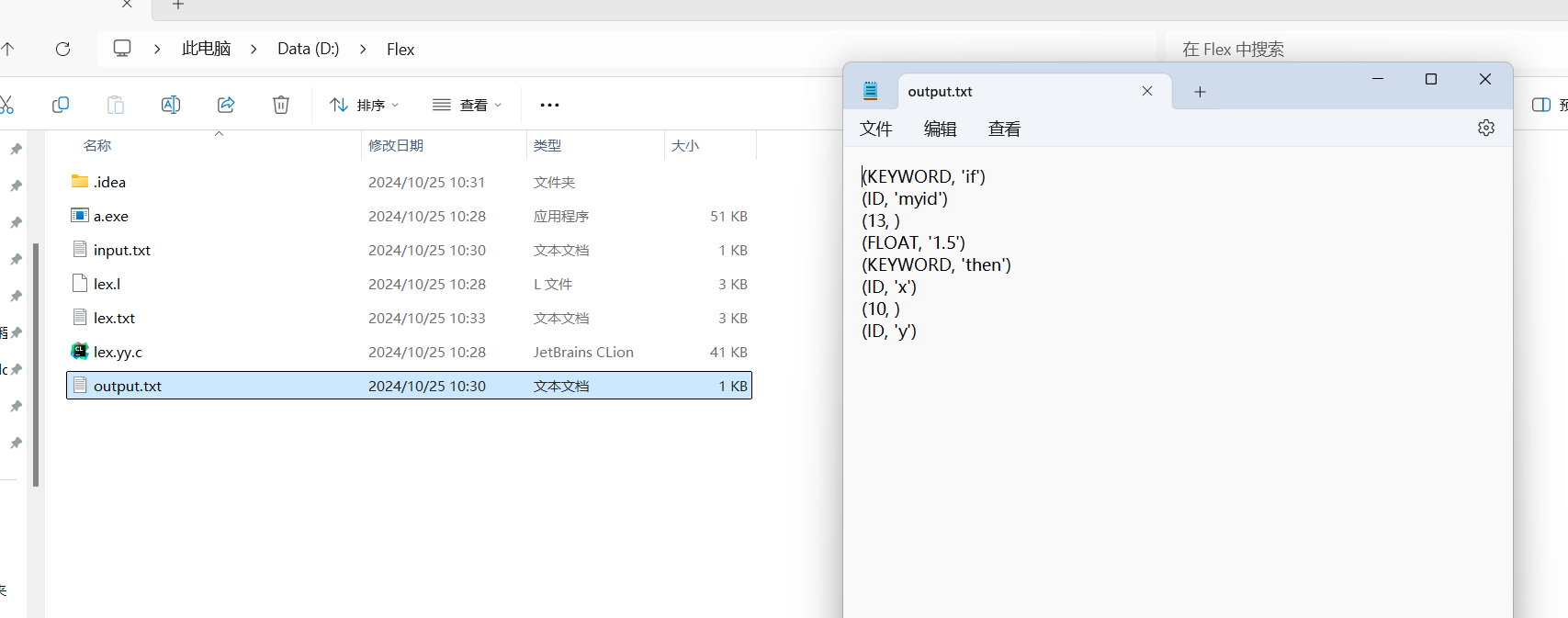


之后以管理员身份打开命令提示符，转到lex.l所在的位置，之后输入flex lex.l，再输入

gcc lex.yy.c，会生成一个a.exe文件，之后再输入a并按下回车，则会发现已生成了output.txt文件：



Output.txt的内容如下：



---------------------------------------------------------------------------------------------------------------------------------

**2、扩展部分：**

在此基础上，我选择完成扩展实验1的内容，其要求如下：

1、扩充关键字的数目、增加逻辑运算符等单词类别、将常数再细分成字符串

常量、整型常量和无符号数常量等；添加词法分析中单词出错的位置和错

误类型，以及删除注释部分等。

于是，我分别在以下这几方面对源代码进行了修改：

**1、增加的符号类型：**

增加了多个符号类型，包括：

* UNSIGNED\_INT：无符号整数
* STRING\_CONST：字符串常量
* BOOL：布尔值
* AND、OR、NOT：逻辑运算符
* 关系运算符（如 GE、LE、LT、GT、NE 等）均有定义。

**2、更复杂的词法规则：**

· **字符串常量**：通过 \"([^\\\"\n]|\\.)\*\" 识别的字符串。

· **逻辑运算符**：&&、||、! 被明确处理为独立的规则。

· **无符号整数**：通过 0-9]+[uU]? 识别，允许标识无符号整数。

**3、处理注释：**

· 实现了对单行注释 // 和多行注释 /\* ... \*/ 的跳过处理。这种处理确保了在分析过程中不会受到注释的影响。

**4、错误处理：**

· 包含了对错误字符的处理。当遇到未定义的字符时，会输出错误信息，指明该字符以及所在行数，提供更好的调试信息。

**5、输出功能的扩展：**

· 新代码中的输出函数 out 增加了对所有新符号类型的处理，使得在输出文件中能够反映更丰富的词法分析结果。

**6、**行号跟踪：****

· 新代码中引入了 line\_number 变量来跟踪行号，这使得在错误处理时能够给出准确的错误行号。

扩展后的代码如下：

%{

    #include <stdio.h>

    #include <stdlib.h>

    #include <string.h>

    #define ID 6

    #define INT 7

    #define LT 8

    #define LE 9

    #define EQ 10

    #define NE 11

    #define GT 12

    #define GE 13

    #define FLOAT 14

    #define STRING\_CONST 15

    #define UNSIGNED\_INT 16

    #define BOOL 17

    #define AND 18

    #define OR 19

    #define NOT 20

    FILE \*output\_fp;

    int line\_number = 1;

    int lookup(char \*token) {

        if (strcmp(token, "if") == 0) return 1;

        if (strcmp(token, "then") == 0) return 2;

        if (strcmp(token, "else") == 0) return 3;

        if (strcmp(token, "begin") == 0) return 4;

        if (strcmp(token, "end") == 0) return 5;

        if (strcmp(token, "true") == 0) return BOOL;

        if (strcmp(token, "false") == 0) return BOOL;

        return 0;

    }

    void out(int code, char \*val) {

        if (code == ID) {

            fprintf(output\_fp, "(ID, '%s')\n", val);

        } else if (code == INT) {

            fprintf(output\_fp, "(INT, '%s')\n", val);

        } else if (code == FLOAT) {

            fprintf(output\_fp, "(FLOAT, '%s')\n", val);

        } else if (code == STRING\_CONST) {

            fprintf(output\_fp, "(STRING\_CONST, '%s')\n", val);

        } else if (code == UNSIGNED\_INT) {

            fprintf(output\_fp, "(UNSIGNED\_INT, '%s')\n", val);

        } else if (code == BOOL) {

            fprintf(output\_fp, "(BOOL, '%s')\n", val);

        } else if (code >= 1 && code <= 5) {

            fprintf(output\_fp, "(KEYWORD, '%s')\n", val);

        } else if (code == AND) {

            fprintf(output\_fp, "(AND, '&&')\n");

        } else if (code == OR) {

            fprintf(output\_fp, "(OR, '||')\n");

        } else if (code == NOT) {

            fprintf(output\_fp, "(NOT, '!')\n");

        } else if (code == GE) {

            fprintf(output\_fp, "(GE, '>=')\n");

        } else if (code == EQ) {

            fprintf(output\_fp, "(EQ, '=')\n");

        } else if (code == LT) {

            fprintf(output\_fp, "(LT, '<')\n");

        } else if (code == LE) {

            fprintf(output\_fp, "(LE, '<=')\n");

        } else if (code == GT) {

            fprintf(output\_fp, "(GT, '>')\n");

        } else if (code == NE) {

            fprintf(output\_fp, "(NE, '!=')\n");

        } else {

            fprintf(output\_fp, "(%d, )\n", code);

        }

    }

    %}

    %%

    [ \t]+                ;

    \n                    { line\_number++; }

    "//".\*                { /\* Ignore single-line comments \*/ }

    "/\*"([^\*]|(\\*+[^\*/]))\*"\\*/"  { /\* Ignore multi-line comments \*/ }

    "if"|"then"|"else"|"begin"|"end"|"true"|"false" {

                            out(lookup(yytext), yytext);

                        }

    "&&"                  { out(AND, "&&"); }

    "||"                  { out(OR, "||"); }

    "!"                   { out(NOT, "!"); }

    \"([^\\\"\n]|\\.)\*\"  { out(STRING\_CONST, yytext); }

    [0-9]+                { out(INT, yytext); }

    [0-9]+[uU]?          { out(UNSIGNED\_INT, yytext); }

    [0-9]+\.[0-9]+       { out(FLOAT, yytext); }

    [a-zA-Z\_][a-zA-Z0-9\_]\* {

                            int code = lookup(yytext);

                            if (code == 0) out(ID, yytext);

                            else out(code, yytext);

                        }

    "<="                  { out(LE, " "); }

    ">="                  { out(GE, " "); }

    "<"                   { out(LT, " "); }

    ">"                   { out(GT, " "); }

    "="                   { out(EQ, " "); }

    "!="                  { out(NE, " "); }

    .                     {

                            fprintf(output\_fp, "Error: Invalid character '%s' at line %d\n", yytext, line\_number);

                        }

    %%

    int yywrap() {

        return 1;

    }

    int main() {

        FILE \*input\_fp;

        input\_fp = fopen("D:\\Flex2\\input.txt", "r");

        if (input\_fp == NULL) {

            perror("Unable to open input file!");

            return 1;

        }

        yyin = input\_fp;

        output\_fp = fopen("D:\\Flex2\\output.txt", "w");

        if (output\_fp == NULL) {

            perror("Unable to open output file!");

            fclose(input\_fp);

            return 1;

        }

        yylex();

        fclose(input\_fp);

        fclose(output\_fp);

        return 0;

    }

编译执行的步骤和之前类似，只不过这次input.txt的内容变成了这个：

if myid >= 1.5 then x = y

// This is a single-line comment

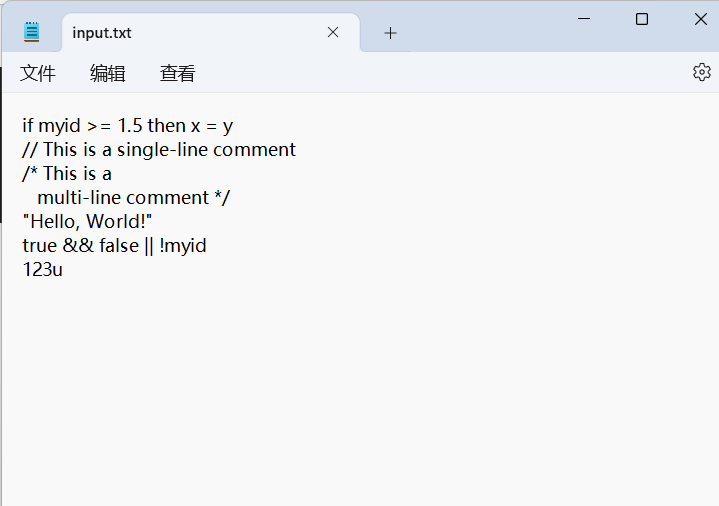
/\* This is a

multi-line comment \*/

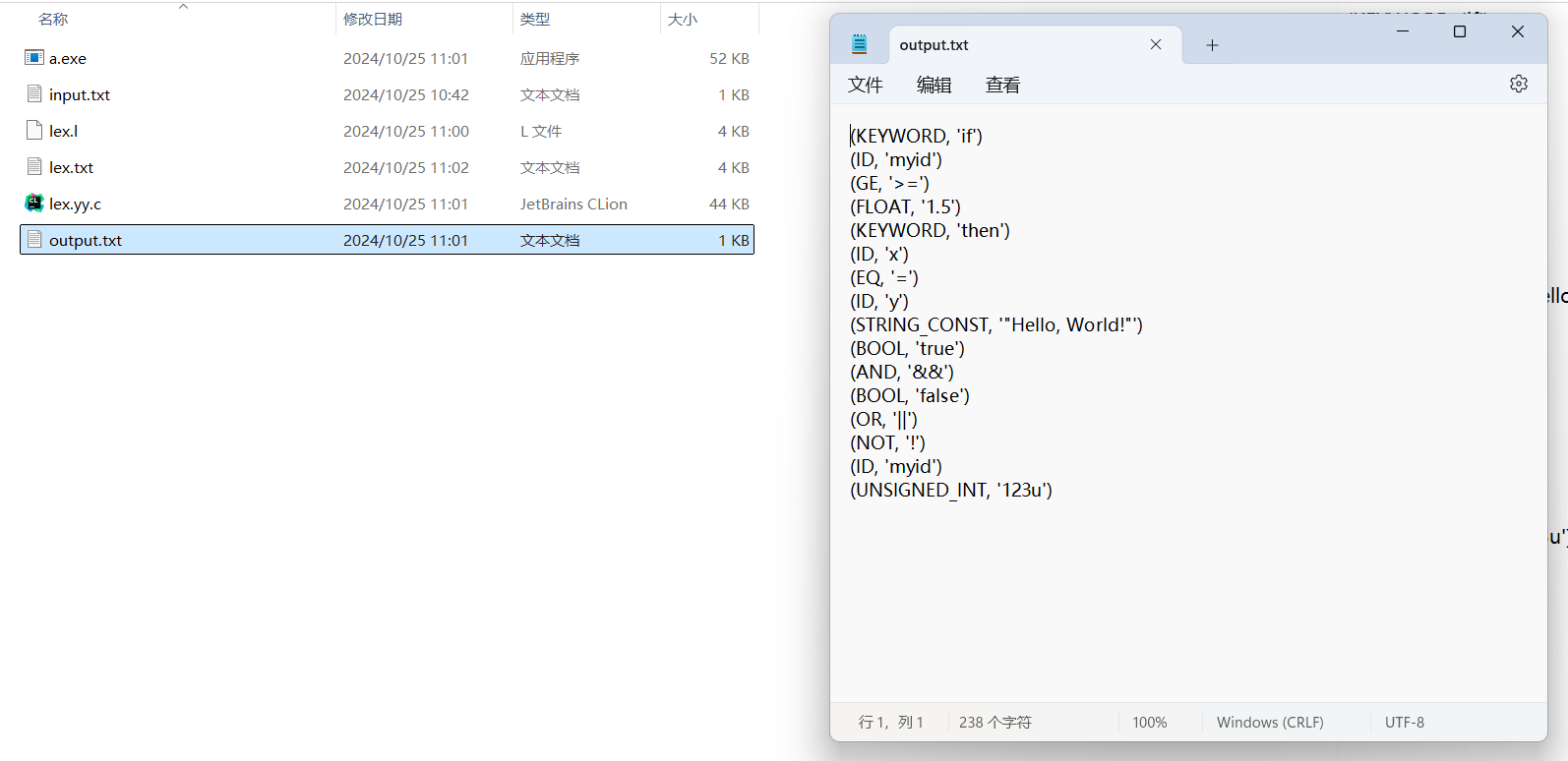
"Hello, World!"

true && false || !myid

123u



Output.txt的结果如下：



---------------------------------------------------------------------------------------------------------------------------------

**3、实验总结：**

本次实验进一步提升了我对词法分析的认识。这次实验的灵感主要来自老师上课时所提到的用到Flex实现词法分析器，因为这种方法对于后续扩充关键字的数目、增加逻辑运算符等单词类别、将常数再细分成字符串常量、整型常量和无符号数常量等操作起来更为方便。正如实验指导书中所说：“开发一种新语言时，由于它的单词符号在不停地修改，采用 LEX 等工具生成的词法分析程序比较易于修改和维护。”本次实验我也有点儿遗憾，就是没能找到怎么用Clion或Vscode上执行，只能用命令提示符的方式编译执行，出现编译错误的修改起来极其不方便。