实验题目和要求

识别并输出单词符号：

- 程序可以识别出 C 语言源程序中的每个单词符号，并以特定格式输出每个单词符号的详细信息。
- 输入文件：程序读取一个名为 `input.txt` 的文件。
- 输出文件：程序将分析结果写入 `output.txt` 中，内容包括每个单词符号的类别、属性和行数。例如，可以输出类似 `<记号，属性，行数>` 的格式。
- 能够识别所有的单词符号，并输出信息。例如，标识符、关键字、操作符、数字常量等。

跳过注释：

- 程序应识别并跳过源程序中的注释，不将注释内容作为单词符号输出。

统计信息：

- 程序可以统计源程序中的总行数、各类单词符号的个数、以及总的字符数，并将这些统计结果输出到文件中。
- 字符数应包括换行符、空格、注释等。

词法错误检查：

- 程序应能够检查源程序中的词法错误，并报告错误所在的行数和错误类型。
- 错误信息需详细描述错误的性质，例如：`ERROR(错误编号): 错误位置，错误说明`。

错误处理和继续分析：

- 程序应具备一定的错误恢复能力，在出现错误后，程序应能进行合适的错误处理，避免因错误中断词法分析。
- 对错误进行适当的恢复，以便能够继续分析源代码。

- 词法分析器的工作内容：将源代码转换为词法单元，识别关键字，操作符，数字、标识符和分隔符。
- 词法分析处理的错误：非法字符、非法标识符，未闭合的字符串或者字符常量、数字格式错误、注释错误。
- 词法分析不需要关注括号的检查
- 对于负数一般是将负号单独识别一个token
- \0一般在编译的语义分析或中间代码生成阶段添加的，词法分析器看到的代码没有\0

程序设计说明

词法分析器功能

- 词法分析的工作内容：将源代码转换为词法单元，识别关键字，操作符，数字、标识符和分隔符。
- 词法分析需要处理的错误：非法字符、非法标识符，未闭合的字符串或者字符常量、数字格式错误、注释错误。

词法分析的token设计以及识别

- 预处理指令

  通常以`#`开头，预处理指令的执行发生在词法分析之前。所以在进行词法分析时，预处理指令已经执行完成并实现了必要的替换。这里我们只是做一个简单的识别，因为实际上词法分析器不会分析预处理指令。

```
#include
#define
#ifdef
#if
#else
#endif
#error
#pragma
#undef
#line
```

- 关键字

  C语言定义了32个关键字，关键字是C语言中预定义的保留字，它们都有特殊的用途，不能将它们用于程序中的标识符。

```
char str[MAX][10]={"int","float","double","char","void",
                   "short","long","signed","unsigned",
                   "struct","union", "enum","typedef","sizeof",

 "auto","static","register","extern","const","volatile",
                   "return","continue","break","goto",
                   "if","else","switch","case","default",
                   "for","while","do"
};
```

  `关于main`:main虽然不是关键字而属于标识符，但是main是在C语言标准中唯一被指定未程序的入口点的函数，是程序的起点。

- 标识符

  标识符由**字母**（A-Z，a-z）、**数字**（0-9）和**下划线**（`_`）组成，且不能以数字开头，而且不能是C语言的关键字(这一条在编译器的语法分析时进行处理，词法分析阶段是可以识别的)。

  标识符主要是四大类

  - 变量标识符
  - 函数标识符

    - 类型定义标识符
    - 宏标识符
- 常量，

  整数、小数、字符和字符串

```
const int  UnsignedInt=34;
const int  Float=35;
const int  ConstantChar=36;
// 转义字符也属于字符常量
const int  ConstantString=37;
```

- 操作符

  这里我将运算符，分割符，定界符都归为操作符

  对于某些操作符比如'+','/'等等，需要进行超前分析。

- Error

  主要分为9种

  1. 非法字符：源代码中包含不属于C语言字符集的非法字符，基本上就是操作符+字母+数字+一些空白字符和控制字符。
  2. 不完整的注释：多行注释或者单行注释不规范，在词法分析中只关心多行注释的规范问题，单行注释不规范会被识别成操作符'/'
  3. 非法标识符：标识符使用不符合规范
  4. 数值常量错误：非法的浮点数，非法的十六进制数字
  5. 非法字符串常量：缺少双引号
  6. 字符常量错误：缺少单引号
  7. 未知操作符：使用非法操作符
  8. 转义序列错误：使用了C语言不支持的非法转义序列

    - 错误处理不会影响后续的词法分析，只是在错误位置进行标记。

---

token_id/error_id

```
1 ~ 32 Keywords
{"int","float","double","char","void",
"short","long","signed","unsigned",
"struct","union", "enum","typedef","sizeof",
"auto","static","register","extern","const","volatile",
"return","continue","break","goto",
"if","else","switch","case","default",
 "for","while","do"}

 33 Identifier
 34 Unsigned integer
 35 Float
 36 Constant Char
 37 Constant String

 38 ~ 87 Operators
```

```
enum Ops {
    // 算术运算符
    OP_PLUS = 36,           // +
    OP_MINUS,               // -
    OP_MULTIPLY,            // *
    OP_DIVIDE,              // /
    OP_MODULO,              // %

    // 关系运算符
    OP_EQUAL,               // ==
    OP_NOT_EQUAL,           // "!="
    OP_GREATER,             // >
    OP_LESS,                // <
    OP_GREATER_EQUAL,       // >=
    OP_LESS_EQUAL,          // <=

    // 逻辑运算符
    OP_LOGICAL_AND,         // &&
    OP_LOGICAL_OR,          // ||
    OP_LOGICAL_NOT,         // "!"

    // 赋值运算符
    OP_ASSIGN,              // =
    OP_PLUS_ASSIGN,         // +=
    OP_MINUS_ASSIGN,        // -=
    OP_MULTIPLY_ASSIGN,     // *=
    OP_DIVIDE_ASSIGN,       // /=
    OP_MODULO_ASSIGN,       // %=
    OP_BITWISE_AND_ASSIGN,  // &=
    OP_BITWISE_OR_ASSIGN,   // |=
    OP_BITWISE_XOR_ASSIGN,  // ^=
    OP_LEFT_SHIFT_ASSIGN,   // <<=
    OP_RIGHT_SHIFT_ASSIGN,  // >>=

    // 自增和自减运算符
    OP_INCREMENT,           // ++
    OP_DECREMENT,           // --

    // 位运算符
    OP_BITWISE_AND,         // &
    OP_BITWISE_OR,          // |
    OP_BITWISE_XOR,         // ^
    OP_BITWISE_NOT,         // ~
    OP_LEFT_SHIFT,          // <<
    OP_RIGHT_SHIFT,         // >>

    // 条件（三元）运算符
    OP_CONDITIONAL,         // "? :"

    // 指针运算符
    OP_DEREFERENCE,         // *
    OP_ADDRESS_OF,          // &

    // 成员访问运算符
    OP_MEMBER_ACCESS,       // .
    OP_POINTER_MEMBER_ACCESS,// ->
```

```
72
73      // 分隔符
74      OP_COMMA,                // ,
75      OP_SEMICOLON,            // ;
76      OP_PARENTHESES_LEFT,     // (
77      OP_PARENTHESES_RIGHT,    // )
78      OP_BRACES_LEFT,          // {
79      OP_BRACES_RIGHT,         // }
80      OP_BRACKETS_LEFT,        // [
81      OP_BRACKETS_RIGHT,       // ]
82      OP_COLON,                // :
83
84      // 预处理指令标识符
85      OP_PREPROCESSOR_ID       //#
86
87      OP_DOUBLE_QUOTE,          // "
88      OP_SINGLE_QUOTE,          // '
89  };
90   Error id
91   1 Invalid Character
92   2. Unterminated Comment
93   3. Invalid Identifier
94   4. Invalid Numeric Constant
95   5. Invalid String Literal
96   6.Invalid Character Constant
97   7.Unknown Operator
98   8.Invalid Escape Sequence
```

统计信息

- Line,记录行数，注释和预处理指令不计入行数，空行计入行数
- Preprocessor Line： 预处理指令行数
- key_word: 关键字数目
- operator：操作符数目
- error:错误数

换行符，空格，退格等分割符不包含在统计结果中。