
# 词法分析器设计说明

学号：1952650

姓名：陈子翔

# 目 录

# 词法分析器题目描述

要求:

1. 对给定类 C 语言的单词集，试编写一个词法分析器，输入为源程序字符串，输出为单词的机内表示序列（<种别，属性值>）。
2. 词法分析器设计为子程序以供主程序或后续饰演的语法分析程序调用。
3. 在完成以上基本要求情况下，对程序功能扩充（如：增加单词数量、出错处理、预处理程序等）。

类 C 词法规则:

1. 关键字：int | void | if | else | while | return
2. 标识符：字母（字母|数字）*（注：不与关键字相同）
3. 数值：数字（数字）*
4. 赋值号：=
5. 算符：+ | - | * | / | = | == | > | >= | < | <= | !=
6. 界符：；
7. 分隔符：，
8. 注释号：/* */ | //

# 1 程序设计说明

## 1.1 编程语言选择及环境

编程语言：C++

配置环境: 处理器: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz(8 CPUs), ~2.0GHz

内存：8192MB RAM

开发平台：Qt Creator 4.11.1 (Community)

Qt Creator 是跨平台的 Qt IDE。此 IDE 能够跨平台运行, 支持的系统包括 Linux(32 位及 64 位)、Mac OS X 以及 Windows。能够使用强大的 C++ 代码编辑器可快速编写代码。

运行环境：Desktop Qt 5.14.2 MinGW 64-bit

## 1.2 输入方式

本程序采用了 Qt 开发平台设计了可视化界面，用户可以在程序运行后通过点击“读入源代码”按钮选择相应的要读入的源代码 txt 文件，按钮下方的文本输入框会将 txt 文件中的代码呈现出来。同时，用户也可直接在该按钮下方的文本框内直接输入要执行词法分析的代码。

读入待分析源程序后，用户可按下“词法分析”按钮展开分析，后续程序会将进行预处理后的源程序以及词法分析的结果打印在文本输入框内。

### 1.3 具体流程

#### 1.3.1 程序预处理

目的：编译预处理，剔除无用的字符和注释。
1.若为单行注释“//”,则去除注释后面的东西，直至遇到回车换行;
2.若为多行注释“/* 。。。*/”则去除该内容;
3.若出现无用字符，则过滤；否则加载；最终产生净化之后的源程序。

#### 1.3.2 词法分析程序实现

1.读入经程序与处理后的源代码并依次对输入的字符进行扫描;
2.首先判断是否是空格、tab 和换行符：忽略空格和 tab，若遇到换行符，行数加 1;
3.首先通过符号是否以字母、下划线开头判断是否是标识符。读入后先与已有关键字一一匹配，若匹配成功，则为关键字，写入 table 表中；若匹配不成功，则不是关键字，按照变量标识符处理;
4.若不以字母、下划线为开头，则判断开头是否是数字，若是则表示遇到数字：区分小数和整数，同时判断是否出错;
5.最后来判断是否是运算符，运算符首先判断双目运算符，再行判断单目运算符。

#### 1.3.3 出错处理

具体实现思路在于按照一定的判别顺序判断读入的信息是否是关键字、标识符、数字、符号；若这些均无法识别，则将读入的信息判别为错误。在此大基础上，在各自判断类别内特判一些个别错误（如开头读入数字，判断后续是否还出现字符等），具体能够识别的错误包括不限于：
1.变量的长度过长（超过 20 位），出现错误;
2.常量的长度过长（超过 20 位），出现错误;
3.表示小数时小数点个数大于 1 个（如 1..1），出现错误;
4.程序中出现无法识别的非法字符;
5.以数字开头，后续出现非数字的其他字符;
6.等。

## 2 程序功能介绍

### 2.1 单词符号与种别码

| 单词符号 | 类型编码 | 单词符号 | 类型编码 |
|---|---|---|---|
| int | 0 | delete | 31 |
| void | 1 | public | 32 |
| if | 2 | struct | 33 |
| else | 3 | 标识符 | 34 |

| | | | |
|---|---|---|---|
| while | 4 | 整数 | 35 |
| return | 5 | 小数 | 36 |
| signed | 6 | <= | 37 |
| char | 7 | >= | 38 |
| double | 8 | := | 39 |
| unsigned | 9 | <> | 40 |
| const | 10 | == | 41 |
| goto | 11 | != | 42 |
| for | 12 | + | 43 |
| float | 13 | - | 44 |
| break | 14 | * | 45 |
| class | 15 | / | 46 |
| case | 16 | = | 47 |
| do | 17 | < | 48 |
| long | 18 | > | 49 |
| typedef | 19 | ( | 50 |
| static | 20 | ) | 51 |
| friend | 21 | , | 52 |
| new | 22 | ; | 53 |
| enum | 23 | . | 54 |
| try | 24 | [ | 55 |
| short | 25 | ] | 56 |
| continue | 26 | : | 57 |
| sizeof | 27 | { | 58 |
| switch | 28 | } | 59 |
| private | 29 | “ | 60 |
| catch | 30 | | |

## 2.2 程序预处理功能

预处理子程序任务:

1.剔除编辑用字符，如 space，tab，CR，LF。

2.剔除注释语句。

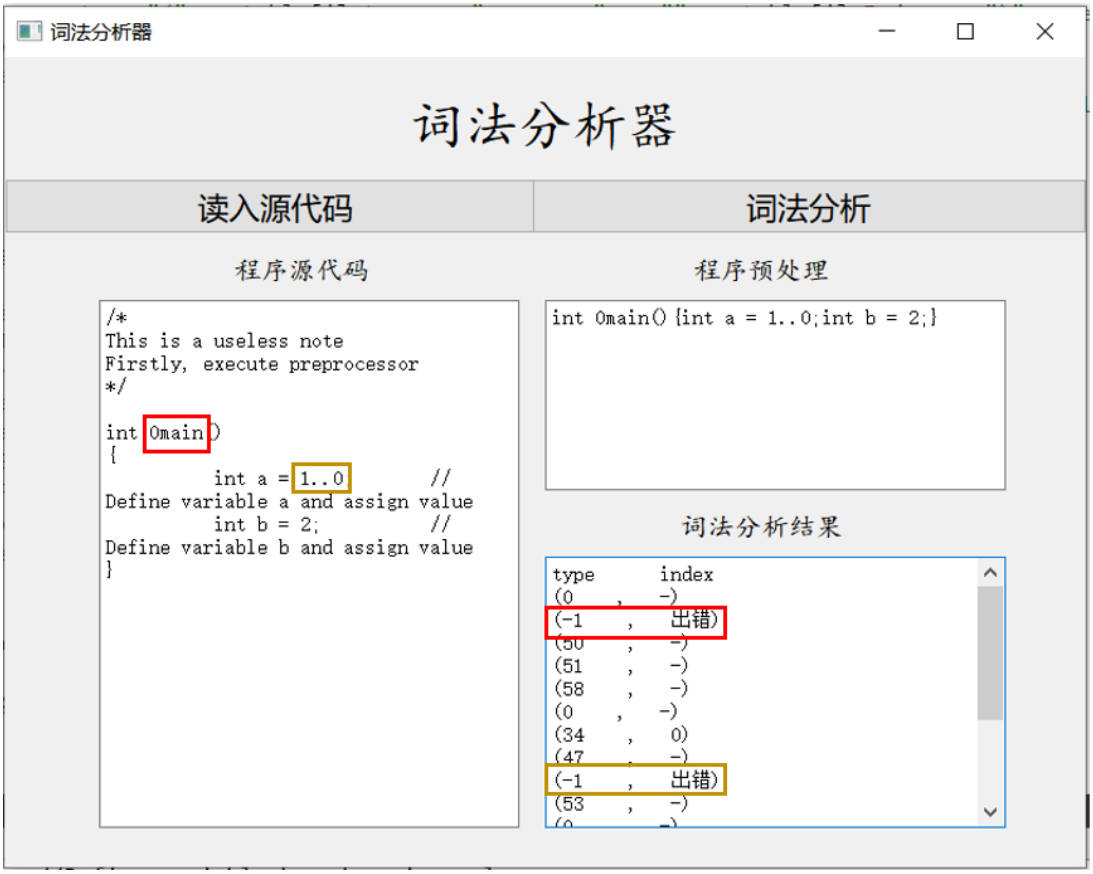
3.合并空白字符串为单个 space。

4.有时把独立文件中的多个源程序模块聚合在一起。

5.把宏定义的缩写形式转换为源语句。


如图执行程序预处理功能。

## 2.3 错误处理




如图可得，此源代码出现了两个错误：“0main”和“1..0”，处理方案为直接将无法识别的字符串（即视为错误）的种别码定为-1，在 index 处输出“出错”，表示此处词法分析出错。

# 3 模块实现原理

## 3.1 词法分析器基本功能实现原理

1.读入经程序与处理后的源代码并依次对输入的字符进行扫描；

```
//对输入的字符扫描
while (chIndex < InputStrLen)
```

2.首先处理空格、tab 和换行符：读到这些字符后，直接忽略；

```
/*处理空格和tab*/
while (InputStr[chIndex] == ' ' || InputStr[chIndex] == 9) //忽略空格和tab
    chIndex++;

/*处理换行符*/
while (InputStr[chIndex] == 10) //遇到换行符，行数加1
{
    Line++;
    chIndex++;
}
```

3.若以字母、下划线开头，则视为标识符执行后续判断：读入后先与已有关键字一一匹配，若匹配成功，则为关键字，写入 table 表中；若匹配不成功，则不是关键字，按照变量标识符处理；

```
/*标识符*/
if (isalpha(InputStr[chIndex])) //以字母、下划线开头
```

```
if (strcmp(str, KeyWord[i]) == 0) //是关键字，写入table表中
```

```
if (i >= MaxKeyWord) //不是关键字
```

4.若不以字母、下划线为开头，则判断开头是否是数字，若是则表示遇到数字：区分小数和整数，同时判断是否出错；

```
/*常数*/
else if (isdigit(InputStr[chIndex])) //遇到数字
```

```
if (InputStr[chIndex] == '.') // flag表记小数点的个数，0时为整数，1时为小数，2时出错
    flag++;
```

5.最后来判断是否是运算符，运算符首先判断双目运算符，再行判断单目运算符。

```
for (i = 0; i < MaxBinaryOptNum; i++) // MaxOptBNum)
    if (strcmp(str, OptB[i]) == 0)
```

OptB 中存放的是二元运算符，首先判断该运算符是否为二元运算符。

```
if (i >= MaxBinaryOptNum)
{
    for (int k = 0; k < MaxUnaryOptNum; k++)
        if (OptA[k] == InputStr[chIndex])
```

若匹配均不成功，则不为二元运算符；再行判断是否为一元运算符。

### 3.2 扩充功能：预处理程序实现原理

目的：编译预处理，取出无用的字符和注释。

1.将源程序所有信息读入输入缓冲区内，并依次扫描缓冲区内的字符；

```
for (int i = 0; i <= InputStrLen; i++)
```

2.若为单行注释“//”,则去除注释后面的东西，直至遇到回车换行；

```
if (InputStr[i] == '/' && InputStr[i + 1] == '/')
{//若为单行注释“//”,则去除注释后面的东西，直至遇到回车换行
    while (InputStr[i] != '\n')
    {
        i++;//向后扫描
    }
}
```

3.若开始读到“/*”则继续扫描，直至读到“*/”，过滤掉内部信息；

```
if (InputStr[i] == '/' && InputStr[i + 1] == '*')
{//若为多行注释“/* 。。。*/”则去除该内容
```

4.若出现无用字符，则过滤；否则加载；

```
if (InputStr[i] != '\n' && InputStr[i] != '\t' && InputStr[i] != '\v' && InputStr[i] != '\r')
{//若出现无用字符，则过滤；否则加载
    tempString[count++] = InputStr[i];
}
```

5.产生净化之后的源程序。

```
    tempString[count] = '\0';
    strcpy(InputStr, tempString);//产生净化之后的源程序
```

### 3.3 扩充功能：出错处理实现原理

1.按照一定的判别顺序判断读入的信息是否是关键字、标识符、数字、符号；若这些均无法识别，则将读入的信息判别为错误；

```
/*其他无法识别字符*/
if (errorFlag != 0 && errorFlag != 1) //开头的不是字母、数字、运算符、界符
{
    char str[256];
    int strLen = -1;
    str[strLen++] = InputStr[chIndex];
    chIndex++;
```

2.若常量/变量标识符超过规定长度 20，报错；

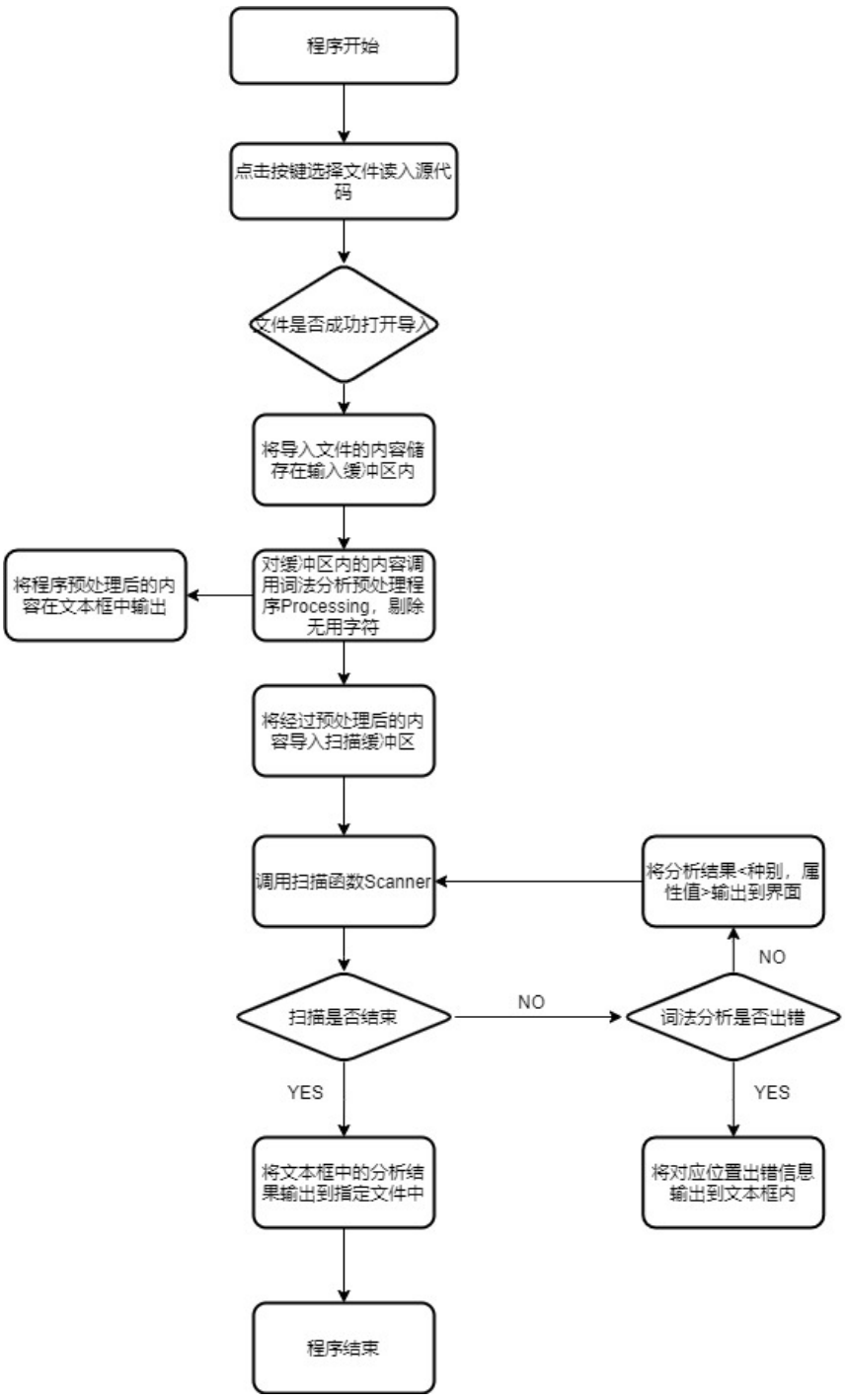
```
if (strlen(str) > 20) //常量标识符超过规定长度20，报错处理
{
    table[TableNum].type = -1; //出现错误

    strcpy(table[TableNum].symbol, "too long");
    table[TableNum].line = Line;
    TableNum++;
    error(str, Line, 3);
}
```

3.若开头为数字，则计算小数点个数，若小数点个数大于 1，则出错。

```
while (isdigit(InputStr[chIndex]) || InputStr[chIndex] == '.') //数字和小数点
{
    if (InputStr[chIndex] == '.') // flag表记小数点的个数，0时为整数，1时为小数，2时出错
        flag++;
    str[strLen++] = InputStr[chIndex];
    chIndex++;
}
```

## 4 算法分析框图

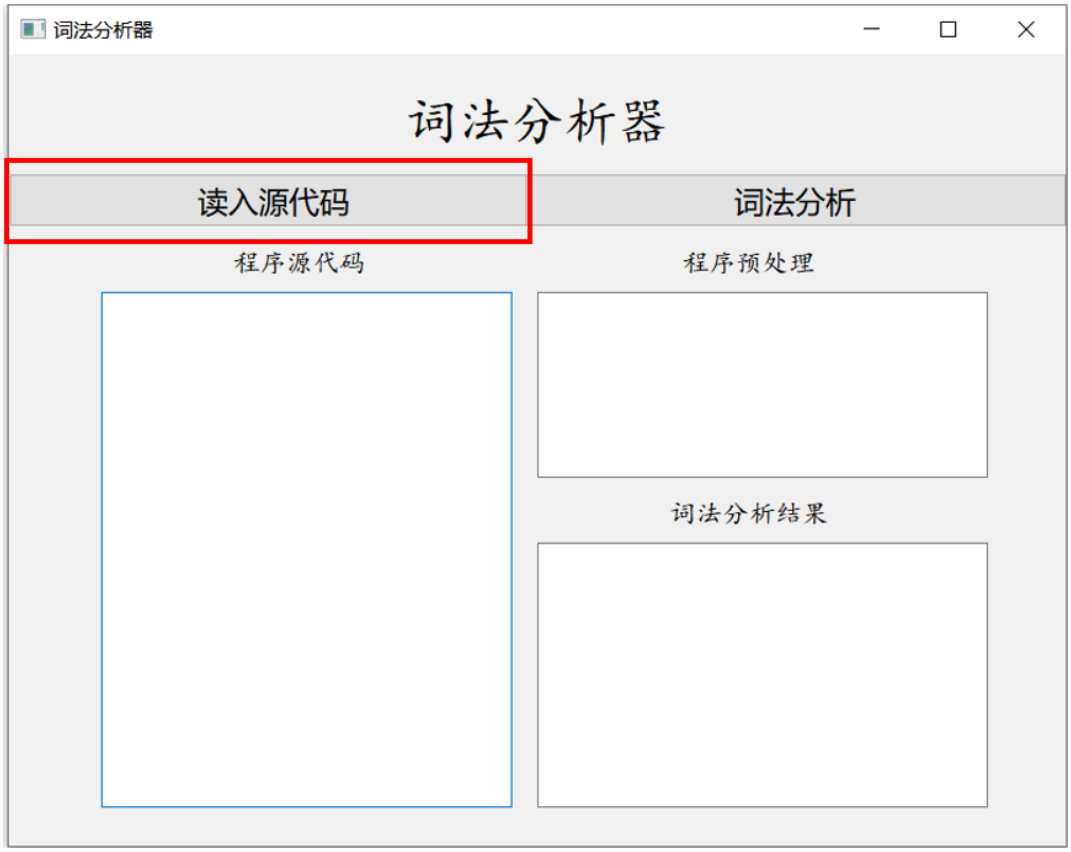


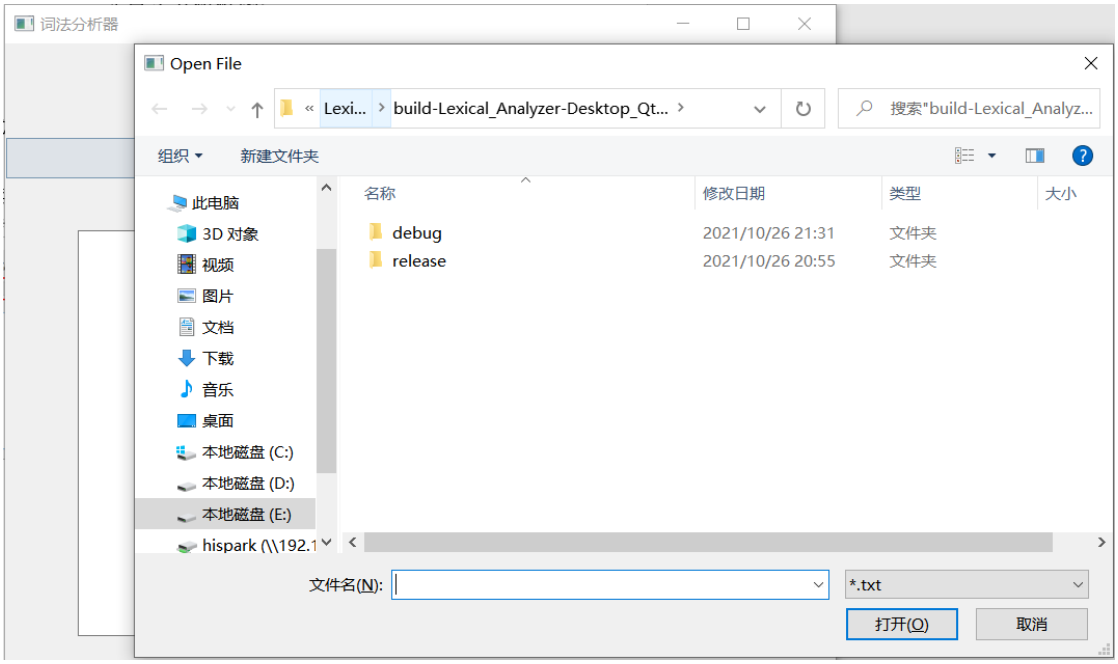
# 5 程序使用说明

## 5.1 输入输出格式

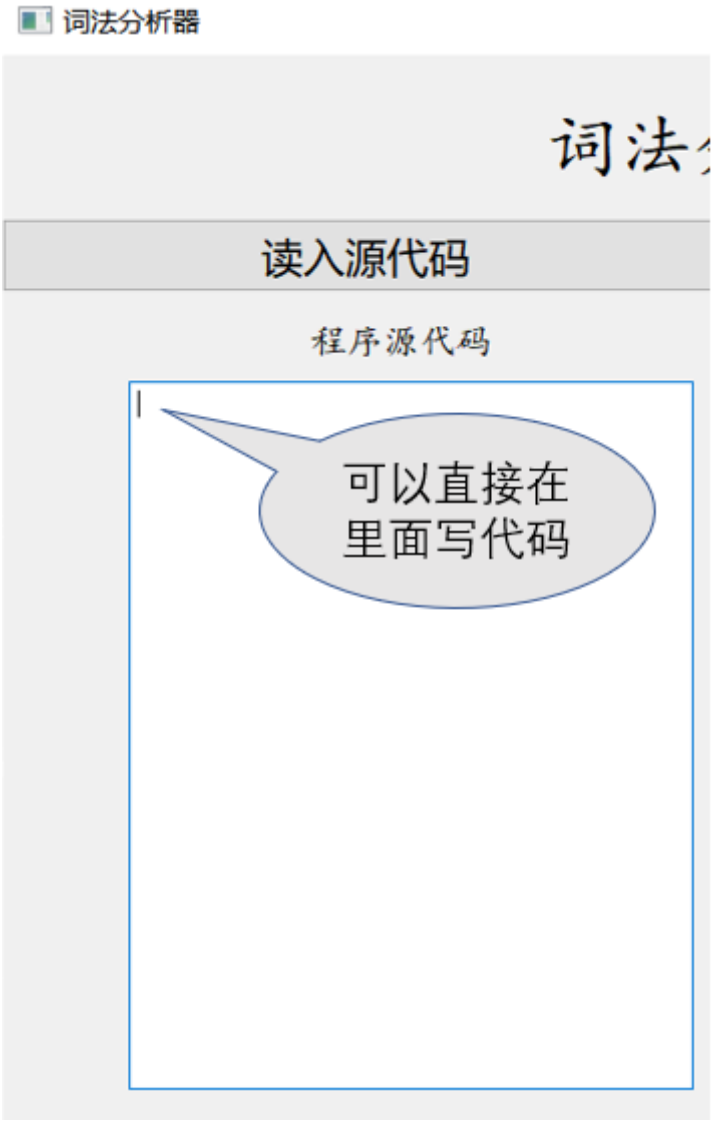
1.输入方式:

① 通过点击“读入源代码”按键导入源代码文件，程序会将文件中的代码显示在文本框中。



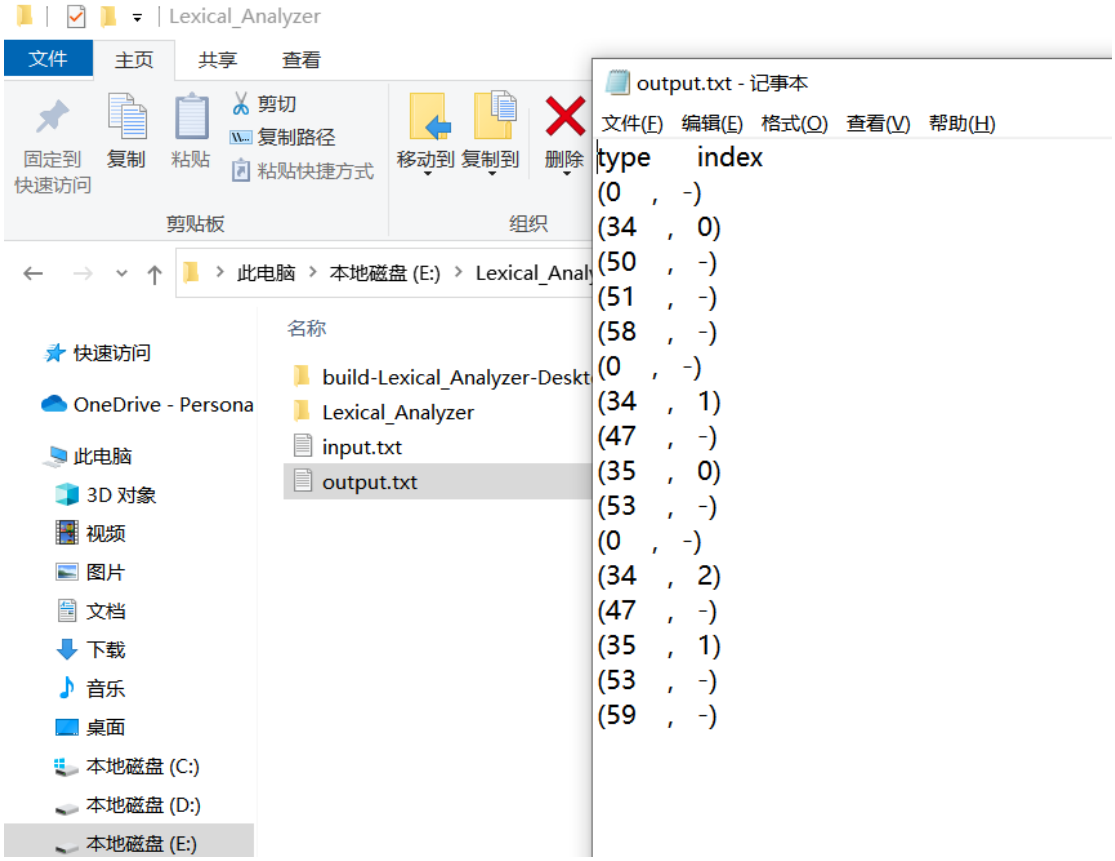

② 还可通过直接在文本框中输入要进行词法分析的源代码进行分析。




2.输出方式:

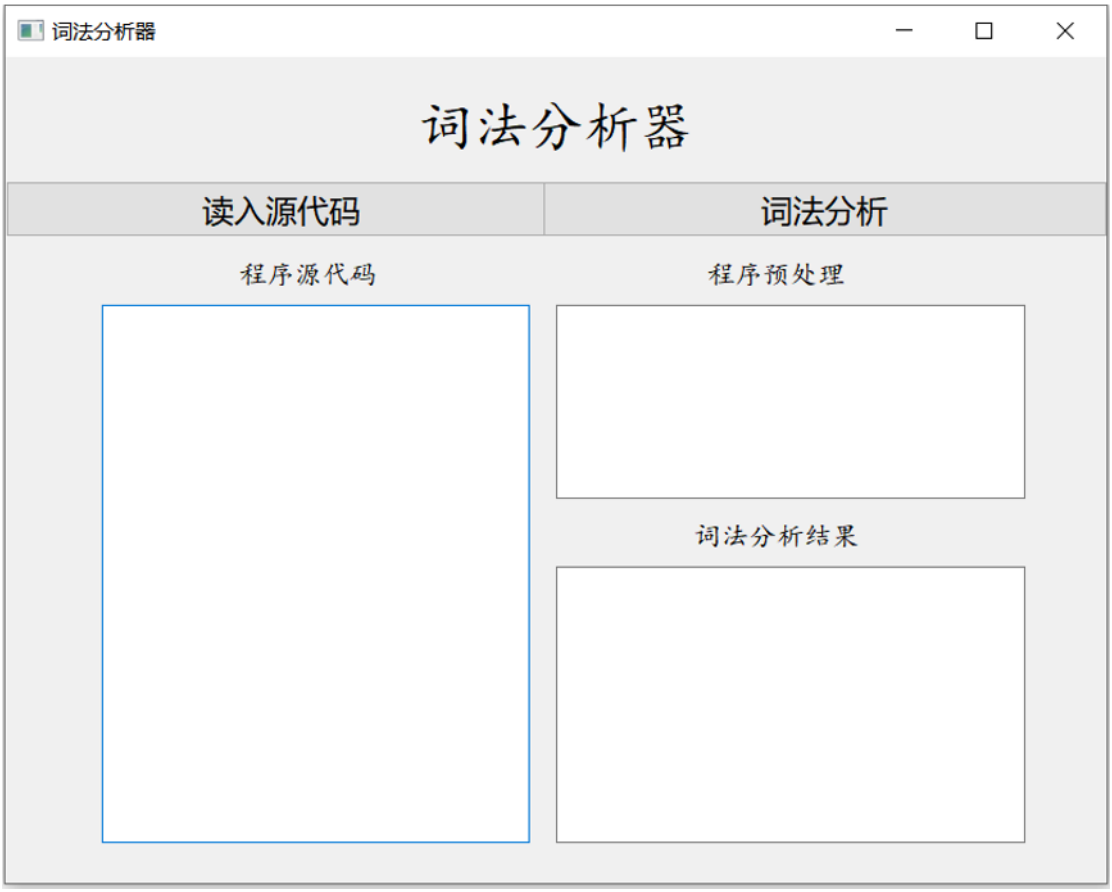
① 程序将经过词法分析所得的结果在屏幕中文本框中打印出来。


② 同时程序将词法分析的结果输出到项目目录下的 output.txt 文本框中。



# 6 执行实例

## 6.1 用户友好界面



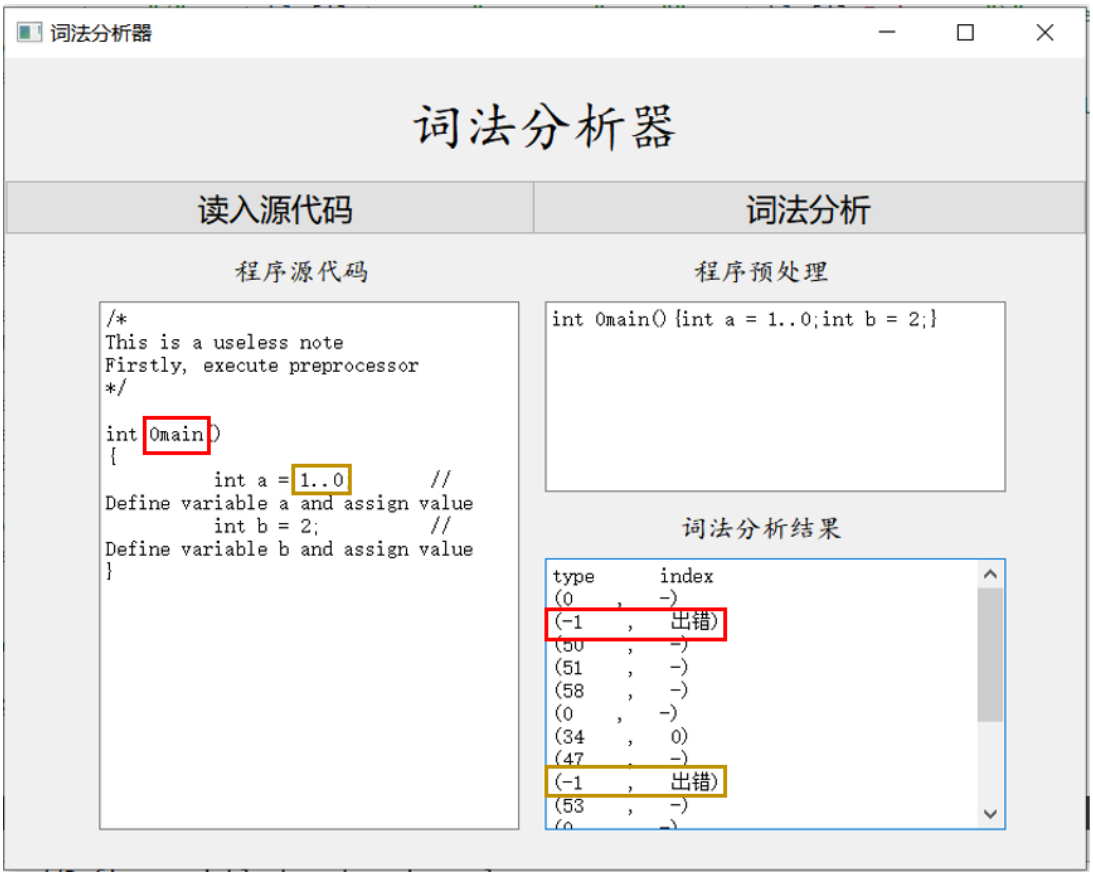

本次词法分析结果并不采用控制台输出，而是采用了交互性更强、更友好的界面。运行程序后，用户可以通过点击“读入源代码”按钮选择要读入源文件的路径并将程序内容输出到左侧“程序源代码”文本框中，直观显示。读入程序内容结束后，点击“词法分析”按钮后，首先会在“程序预处理”文本框内输出经过预处理后的源代码，再在“词法分析结果”文本框内输出程序的词法分析结果。总体交互体验不错。

## 6.2 运行结果截图

无注释情况下源代码:


有注释情况下源代码（程序预处理）：


代码有错误情况下词法分析程序的处理:



如图可得，此源代码出现了两个错误：“0main”和“1..0”，处理方案为直接将无法识别的字符串（即视为错误）的种别码定为-1，在 index 处输出“出错”，表示此处词法分析出错。

## 7 程序源代码

```
    #include "widget.h"
#include "ui_widget.h"

#include <QTextStream>
#include <QDateTime>
#include <QFile>
#include <QDebug>
#include <QFileDialog>
#include <QTextCodec>
#include <QPushButton>
#include <string>

//QIODevice::NotOpen    未打开
//QIODevice::ReadOnly   以只读方式打开
//QIODevice::WriteOnly 以只写方式打开
//QIODevice::ReadWrite 以读写方式打开
//QIODevice::Append     以追加的方式打开，新增加的内容将被追加到文件末尾
//QIODevice::Truncate   以重写的方式打开，在写入新的数
//QIODevice::Text       在读取时，将行结束符转换成 \n；在写入时，将行结束符转换成本地格式，例如 Win32 平台上是 \r\n
//QIODevice::Unbuffered       忽略缓存

#define MaxCodeLength 655           //最大代码长度
#define WordMaxNum 256              //变量最大个数
#define DigitNum 256                //常量最大个数
#define MaxKeyWord 34               //关键字数量
#define MaxUnaryOptNum 7            //单目运算符最大个数
#define MaxBinaryOptNum 6           //双目运算符最大个数
#define MaxEndNum 11                //界符最大个数

enum errorType
{
    VarExceed = 1,
    PointError = 2,
    ConExceed = 3
};
```

```
typedef struct DisplayTable
{
    int Index;       //标识符所在表的下标
    int type;        //标识符的类型
    int line;        //标识符所在表的行数
    char symbol[20]; //标识符所在表的名称
} Table;

int TableNum = 0;           //display 表的下标
char Word[WordMaxNum][20];  //标识符表
char Digit[WordMaxNum][20]; //数字表
int WordNum = 0;            //变量表的下标
int DigNum = 0;             //常量表的下标
bool errorFlag = 0;         //错误标志

const char *const KeyWord[MaxKeyWord] = {"int", "void", "if", "else",
"while", "return",
                                         "signed", "char", "double",
"unsigned", "const",
                                         "goto", "for", "float",
"break", "class", "case",
                                         "do", "long", "typedef",
"static", "friend",
                                         "new", "enum", "try",
"short", "continue", "sizeof",
                                         "switch", "private",
"catch", "delete", "public", "struct"};
const char OptA[] = {'+', '-', '*', '/', '=', '<', '>'}; // 单目运算
const char *OptB[] = {"<=", ">=", ":=", "<>", "==", "!="};
//双目运算符
const char End[] = {
    '(', ')', ',', ';', '.', '[',
    ']', ':', '{', '}', '"'}; // 界符

/*
0——33：关键字："int", "void", "if", "else", "while", "return"
34：标识符
35：整数
36：小数
37——42：双目运算符："<=", ">=", ":=", "<>", "==", "!="
43——49：单目运算符：'+', '-', '*', '/', '=', '<', '>'
```

```
50———60: '(', ')', ',', ';', '.', '[', ']', ':', '{', '}', '"'
*/

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    setWindowTitle("词法分析器");


QObject::connect(ui->writeButton,SIGNAL(clicked(bool)),this,SLOT(writ
eButtonSlot()));

QObject::connect(ui->readButton ,SIGNAL(clicked(bool)),this,SLOT( rea
dButtonSlot()));
}

//判断是否为数字
bool isNum(char ch)
{
    if (ch>='0' && ch<='9')
        return true;
    return false;
}

//判断是否为字母
bool isLetter(char ch)
{
    if(ch >= 'a' && ch <= 'z')
        return true;
    return false;
}

//错误处理
void error(char str[20], int nLine, int errorType)
{
    qDebug() << " \nError :    ";
    switch (errorType)
    {
    case VarExceed:
```

```
            qDebug() << "第" << nLine - 1 << "行" << str << " 变量的长度
超过限制! \n";
            errorFlag = 1;
            break;
        case PointError:
            qDebug() << "第" << nLine - 1 << "行" << str << " 小数点错
误! \n";
            errorFlag = 1;
            break;
        case ConExceed:
            qDebug() << "第" << nLine - 1 << "行" << str << " 常量的长度
超过限制! \n";
            errorFlag = 1;
            break;
        }
}

/*********************编译预处理，取出无用的字符和注释
**********************/
void Preprocessing(char InputStr[], int InputStrLen)
{
    char tempString[MaxCodeLength];
    int count = 0;
    for (int i = 0; i <= InputStrLen; i++)
    {
        if (InputStr[i] == '/' && InputStr[i + 1] == '/')
        {//若为单行注释“//”,则去除注释后面的东西，直至遇到回车换行
            while (InputStr[i] != '\n')
            {
                i++;//向后扫描
            }
        }
        if (InputStr[i] == '/' && InputStr[i + 1] == '*')
        {//若为多行注释“/* 。。。*/”则去除该内容
            i += 2;
            while (InputStr[i] != '*' || InputStr[i + 1] != '/')
            {
                i++;//继续扫描
                if (InputStr[i] == '$')
                {
                    printf("注释出错，没有找到 */，程序结束！！！
\n");
```

```
                    exit(0);
                }
            }
            i += 2;//跨过“*/”
        }
        if (InputStr[i] != '\n' && InputStr[i] != '\t' &&
InputStr[i] != '\v' && InputStr[i] != '\r')
        {//若出现无用字符，则过滤；否则加载
            tempString[count++] = InputStr[i];
        }
    }
    tempString[count] = '\0';
    strcpy(InputStr, tempString);//产生净化之后的源程序
}

void Scanner(char InputStr[], int InputStrLen, Table
table[MaxCodeLength], int Line)
{
    int chIndex = 0;

    //对输入的字符扫描
    while (chIndex < InputStrLen)
    {
        /*处理空格和 tab*/
        while (InputStr[chIndex] == ' ' || InputStr[chIndex] == 9) //
忽略空格和 tab
            chIndex++;

        /*处理换行符*/
        while (InputStr[chIndex] == 10) //遇到换行符，行数加 1
        {
            Line++;
            chIndex++;
        }

        /*标识符*/
        if (isalpha(InputStr[chIndex])) //以字母、下划线开头
        {
            char str[256];
            int strLen = 0;
            while (isalpha(InputStr[chIndex]) || InputStr[chIndex] ==
'_') //是字母、下划线
```

```
            {
                str[strLen++] = InputStr[chIndex];
                chIndex++;
                while (isdigit(InputStr[chIndex])) //不是第一位，可以
为数字
                {
                    str[strLen++] = InputStr[chIndex];
                    chIndex++;
                }
            }
            str[strLen] = 0;       //字符串结束符
            if (strlen(str) > 20) //标识符超过规定长度，报错处理
            {
                error(str, Line, 1);
            }
            else
            {
                int i;
                for (i = 0; i < MaxKeyWord; i++)       //与关键字匹配
                    if (strcmp(str, KeyWord[i]) == 0) //是关键字，写
入 table 表中
                    {
                        strcpy(table[TableNum].symbol, str);
                        table[TableNum].type = i; //关键字
                        table[TableNum].line = Line;
                        table[TableNum].Index = i;
                        TableNum++;
                        break;
                    }
                if (i >= MaxKeyWord) //不是关键字
                {

                    table[TableNum].Index = WordNum;
                    strcpy(Word[WordNum++], str);
                    table[TableNum].type = MaxKeyWord; //变量标识符
                    strcpy(table[TableNum].symbol, str);
                    table[TableNum].line = Line;
                    TableNum++;
                }
            }
        }
```

```
        /*常数*/
        // else if(isdigit(ch[chIndex])&&ch[chIndex]!='0') //遇到数字
        else if (isdigit(InputStr[chIndex])) //遇到数字
        {
            int flag = 0;
            char str[256];
            int strLen = 0;
            while (isdigit(InputStr[chIndex]) || InputStr[chIndex] ==
'.') //数字和小数点
            {
                if (InputStr[chIndex] == '.') // flag 表记小数点的个
数，0 时为整数，1 时为小数，2 时出错
                    flag++;
                str[strLen++] = InputStr[chIndex];
                chIndex++;
            }

            str[strLen] = 0;

//            if (InputStr[chIndex] >= 'a' && InputStr[chIndex] <=
'z')
//            {
//                table[TableNum].type = -5; //整数
//                strcpy(table[TableNum].symbol, "too long");
//                table[TableNum].line = Line;
//                TableNum++;
//                chIndex++;
//            }


            if (strlen(str) > 20) //常量标识符超过规定长度 20，报错处
理
            {
                table[TableNum].type = -1; //出现错误

                strcpy(table[TableNum].symbol, "too long");
                table[TableNum].line = Line;
                TableNum++;
                //error(str, Line, 3);
            }
            else if (flag == 0)
            {
```

```
            table[TableNum].type = MaxKeyWord + 1; //整数
        }
        else if (flag == 1)
        {
            table[TableNum].type = MaxKeyWord + 2; //小数
        }
        else if (flag > 1)
        {
            table[TableNum].type = -2; //两个小数点出错

            strcpy(table[TableNum].symbol, "double . error");
            table[TableNum].line = Line;
            TableNum++;
            //error(str, Line, 2);
        }
        if (flag <= 1 && strlen(str) <= 20)
        {
            table[TableNum].Index = DigNum;
            strcpy(Digit[DigNum++], str);

            strcpy(table[TableNum].symbol, str);
            table[TableNum].line = Line;
            TableNum++;
        }
    }

    /*运算符*/
    else
    {
        int errorFlag; //用来区分是不是无法识别的标识符，0 为运算符，1 为界符

        char str[3];
        str[0] = InputStr[chIndex];
        str[1] = InputStr[chIndex + 1];
        str[2] = '\0';
        int i;
        for (i = 0; i < MaxBinaryOptNum; i++) // MaxOptBNum)
            if (strcmp(str, OptB[i]) == 0)
            {
                errorFlag = 0;
                table[TableNum].type = MaxKeyWord + 3 + i;
```

```
                    strcpy(table[TableNum].symbol, str);
                    table[TableNum].line = Line;
                    table[TableNum].Index = i;
                    TableNum++;
                    chIndex = chIndex + 2;
                    break;
                }
            if (i >= MaxBinaryOptNum)
            {
                for (int k = 0; k < MaxUnaryOptNum; k++)
                    if (OptA[k] == InputStr[chIndex])
                    {
                        errorFlag = 0;
                        table[TableNum].type = MaxKeyWord + 3 +
MaxBinaryOptNum + k;
                        table[TableNum].symbol[0] =
InputStr[chIndex];
                        table[TableNum].symbol[1] = 0;
                        table[TableNum].line = Line;
                        table[TableNum].Index = k;
                        TableNum++;
                        chIndex++;
                        break;
                    }

                /*界符*/
                for (int j = 0; j < MaxEndNum; j++)
                    if (End[j] == InputStr[chIndex])
                    {
                        errorFlag = 1;
                        table[TableNum].line = Line;
                        table[TableNum].symbol[0] =
InputStr[chIndex];
                        table[TableNum].symbol[1] = 0;
                        table[TableNum].Index = j;
                        table[TableNum].type = MaxKeyWord + 3 +
MaxBinaryOptNum + MaxUnaryOptNum + j;
                        TableNum++;
                        chIndex++;
                    }
                /*其他无法识别字符*/
```

```
                if (errorFlag != 0 && errorFlag != 1) //开头的不是字
母、数字、运算符、界符
                {
                    char str[256];
                    int strLen = -1;
                    str[strLen++] = InputStr[chIndex];
                    chIndex++;

                    while (*InputStr != ' ' || *InputStr != 9 ||
InputStr[chIndex] != 10) //
                    {
                        str[strLen++] = InputStr[chIndex];
                        chIndex++;
                    }
                    str[strLen] = 0;
                    table[TableNum].type = 1100000;
                    strcpy(table[TableNum].symbol, str);
                    table[TableNum].line = Line;
                    table[TableNum].Index = -2;
                    TableNum++;
                }
            }
        }
    }
}

void Widget::readButtonSlot()
{
    QString fileName = QFileDialog::getOpenFileName(
                    this,
                    tr("Open File"),
                    "",
                    tr("*.txt"));
    QFile file( fileName );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
    QString line = in.readAll();//读取所有
```

```
    ui->textEdit->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}

void Widget::writeButtonSlot()
{
     QString writeData = ui->textEdit->toPlainText();

     char StringkeyTokeys[MaxCodeLength];
     //QString 转 char 数组
     QByteArray ba = writeData.toLocal8Bit();
     memcpy(StringkeyTokeys, ba.data(), ba.size() + 1);
     int nLine = 1;                              //初始化行数
     Table *table = new Table[MaxCodeLength];
     qDebug() << StringkeyTokeys << endl;

     //编译预处理
     Preprocessing(StringkeyTokeys, strlen(StringkeyTokeys));

     ui->textEdit_3->setText(StringkeyTokeys);

     Scanner(StringkeyTokeys, strlen(StringkeyTokeys), table, nLine);
//调用扫描函数

     QFile data("E:/Lexical_Analyzer/output.txt");
     if (data.open( QIODevice::Text | QFile::WriteOnly  ))
     {
         qDebug()<<"yes write";
         QTextStream out(&data);

         qDebug() << "数据量: " << TableNum << endl;
         /*把结果打印到各个 txt 文档中*/

         out << "type" << "       " << "index" << endl;
         for (int i = 0; i < TableNum; i++) //打印 display
         {
             //打印关键字
             if (table[i].type >= 0 && table[i].type <= 5)
                 //out << "(" << table[i].type << "    ,    " <<
table[i].symbol << ")" << endl;
```

```
                out << "(" << table[i].type << "    ,    -" << ")" <<
endl;
            //打印标识符、整数、小数
            else if (table[i].type == 34 || table[i].type == 35 ||
table[i].type == 36)
                out << "(" << table[i].type << "    ,    " << "" <<
table[i].Index << ")" << endl;
            //打印运算符、界符
            else
                //out << "(" << table[i].type << "    ,    " <<
table[i].symbol << ")" << endl;
                out << "(" << table[i].type << "    ,    -" << ")" <<
endl;
        }
    }
    else
       qDebug()<<"no write";

    data.close();
    QFile file("E:/Lexical_Analyzer/output.txt");
    file.open(QIODevice::ReadOnly | QIODevice::Text);
    QByteArray t = file.readAll();
    QString str(t);
    ui->textEdit_2->setText(str);
    //设置文本框只读
    //ui->textEdit_2->setReadOnly(true);
    file.close();
}

Widget::~Widget()
{
    delete ui;
}
```