



燕山大学

编译原理课程设计报告

题目 词法分析器的设计与实现

学 院	<u>信息科学与工程学院</u>
年级专业	<u>2020 级计算机四班</u>
学生姓名	<u>杜钊澎</u>
学生学号	<u>202011040106</u>
设计日期	<u>2022 年 12 月 12 - 16 日</u>

1 概述

1.1 目的与意义

通过设计、编制和调试一个具体的词法分析程序，加深对词法分析原理的理解。掌握在对程序设计语言源程序进行扫描过程中将其分解为各类单词的词法分析方法。

1.2 主要功能

- (1) 编制一个读单词过程，支持 C++ 程序设计语言中 5 类单词的统计分析。
- (2) 说明需要识别的单词的词法规则，并用状态转换图表示。
- (3) 从输入的源程序中，识别出各个具有独立意义的单词，即基本保留字、标识符、常数、运算符、界符五大类。并依次输出各个单词符号自身值、所在行和单词类型。
- (4) 当遇到错误时，指出错误位置和可能的错误原因。准备多组测试样例，对测试样例进行测试和验证，并对输出结果进行分析。

1.3 使用的开发工具

软件、硬件环境：

软件：Visual Studio 2019 编写、调试并执行代码

硬件：笔记本电脑，windows10

1.4 课程设计的计划

周一、周三 2 天腾讯会议线上打卡做课设、上午 8:00-11:00 下午 13:30-16:30，周二全天、周四上午自行查阅资料做课设、撰写报告，周四下午进行验收

1.5 解决的主要问题

在进行五大类字符的词法分析的时候，需要注意自定义标识符和基本保留字的区别，需要将其在判断字母时注意区别，避免产生混淆；

在处理输出整型常量和浮点型常量时，由于不知道如何将所获取的字符串转换成整型和浮点型，因此了解到了可以使用 `atoi` 和 `atof` 函数来将字符串转换为对应的常数类型——虽然表面上解决了这个问题，但是在和老师沟通交流过后，仍旧发现了未从根本上解决这一问题；

在 `CiFa_analysis()` 函数中，起初我在进行结束分析一个字符后，不会将文件指针进行后退，这样可能会影响词法分析的准确性，因此在后来，我增添了

`fseek(fp, -1L, SEEK_CUR)`语句, 来保证在分析完一个字符之后, 能够让文件的指针后退一个字节, 重新读取上一个分析完毕后的字符的后面的字符。

2 状态转换图的基本概念和原理

2.1 基本概念

状态转换图：是设计和实现扫描器的一种有效工具；是一组由矢线连接的有限个结点组成的方向图，其中：每一个结点对应在识别或分析状态中扫描器所处的状态，用小圆圈表示；含有一个初态和若干个终态，分别指示分析的开始和结束；初态用剪头指示，终态用双圆圈表示

2.2 文法

文法 $G[S]$:

$S \rightarrow \text{基本保留字} | \text{运算符} | \text{分界符} | \text{整型常量} | \text{浮点型常量} | \text{标识符}$

基本保留字 $\rightarrow \text{void} | \text{float} | \text{switch} | \text{case} | \text{main} | \text{int} | \text{if} | \text{else} | \text{return} | \text{for} | \text{while}$

运算符 $\rightarrow + | - | * | / | = | < | <= | > | >=$

分界符 $\rightarrow (|) | [|] | \{ | \} | , | ;$

整型常量 $\rightarrow \text{Num}(\text{Num})^*$

浮点型常量 $\rightarrow \text{Num}(\text{Num}).\text{Num}(\text{Num})$

自定义标识符 $\rightarrow \text{letter}(\text{letter} | \text{Num})^*$

$\text{Num} \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

$\text{letter} \rightarrow a | b | \dots | z | A | B | \dots | Z$

2.3 分析原理

通过状态转换图实现对单词的识别。

输入： 符号串

输出： YES/NO — 该符号串是否符合状态转换图的定义；状态转换路径

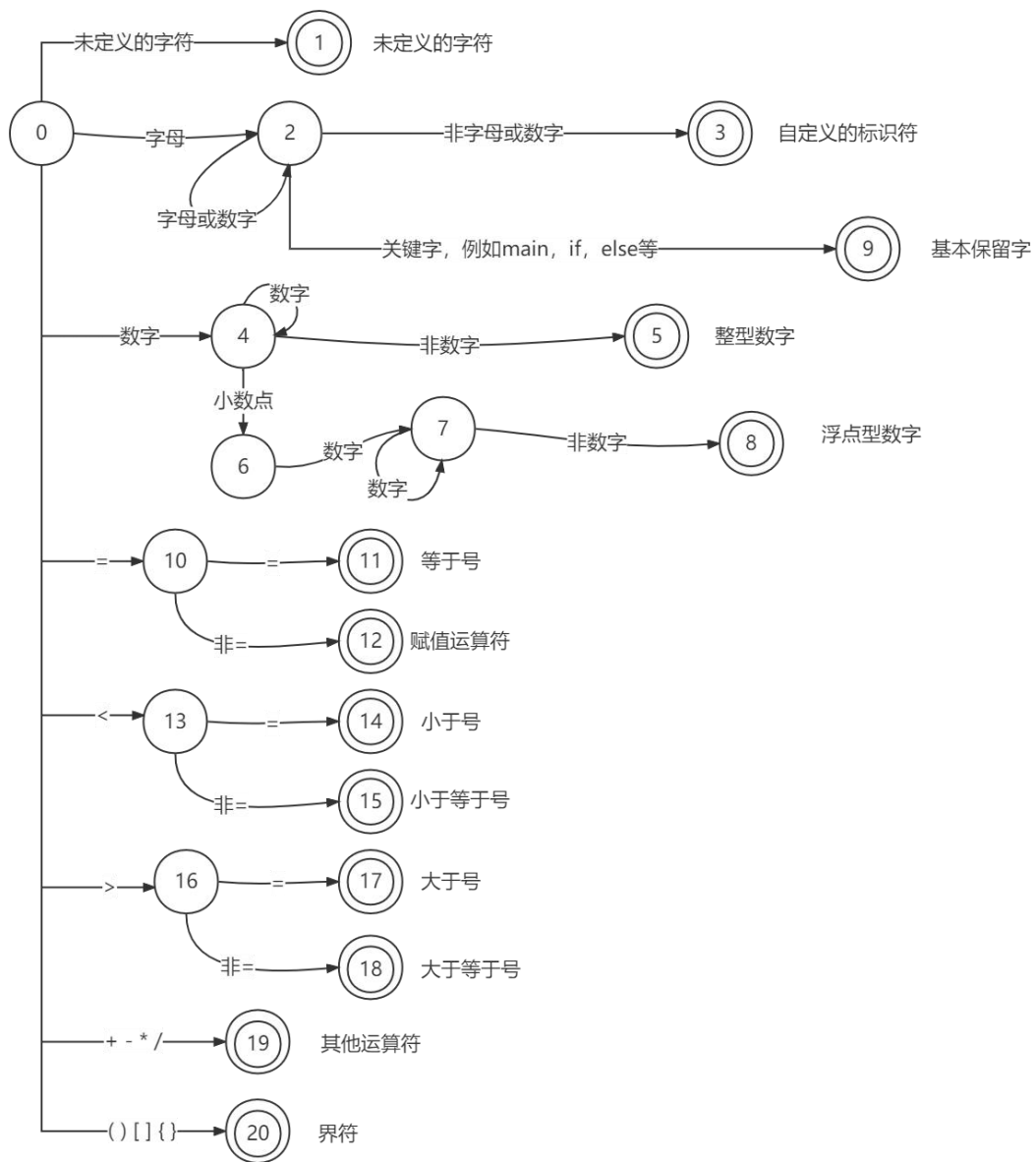


图 2.1 状态转换图

3 总体设计

本项目主要采取的是面向过程的编程方式，通过先将所要判别的单词进行编码，然后通过不同模块分别对文件中的单词进行读取、分析、判定类别、根据类别输出编码及所在位置，最终实现词法分析这一功能。

程序流程如下：

1. 依次读入源程序符号，对源程序进行单词切分和识别，直到源程序结束；读取文件中的符号，对其中的单词进行识别和分析，直至将源文件中的符号全部读取完毕；
2. 对正确的单词，按照它的种别以<种别码，值>的形式保存在符号表中；根据不同的单词类型，将可识别的单词划分成五大类，分别是基本保留字（关键字）、标识符、常数、运算符、界符，并根据这些单词类型的特点进行逐一判别，另外，当识别到“¥”等五大类类型之外的单词，则会将其归为“未识别的单词”；
3. 最终将全部识别结果由第一行至最后一行逐一列出，其展示格式为（类型编码，单词，所在行），并用不同颜色来表示不同类别。

表 3.1 单词分类表

单词符号	种类	编码
未定义的符号	未定义的符号	0
void	基本保留字	1
main	基本保留字	2
int	基本保留字	3
double	基本保留字	4
for	基本保留字	5
while	基本保留字	6
switch	基本保留字	7
case	基本保留字	8
if	基本保留字	9
else	基本保留字	10
return	基本保留字	11

+	运算符	12
-	运算符	13
*	运算符	14
/	运算符	15
=	运算符	16
==	运算符	17
<	运算符	18
<=	运算符	19
>	运算符	20
>=	运算符	21
(界符	22
)	界符	23
[界符	24
]	界符	25
{	界符	26
}	界符	27
,	界符	28
;	界符	29
整形常量	常数	30
浮点型常量	常数	31
自定义标识符	标识符	32

4 详细设计

描述模块内部的流程和实现算法，画出流程图：

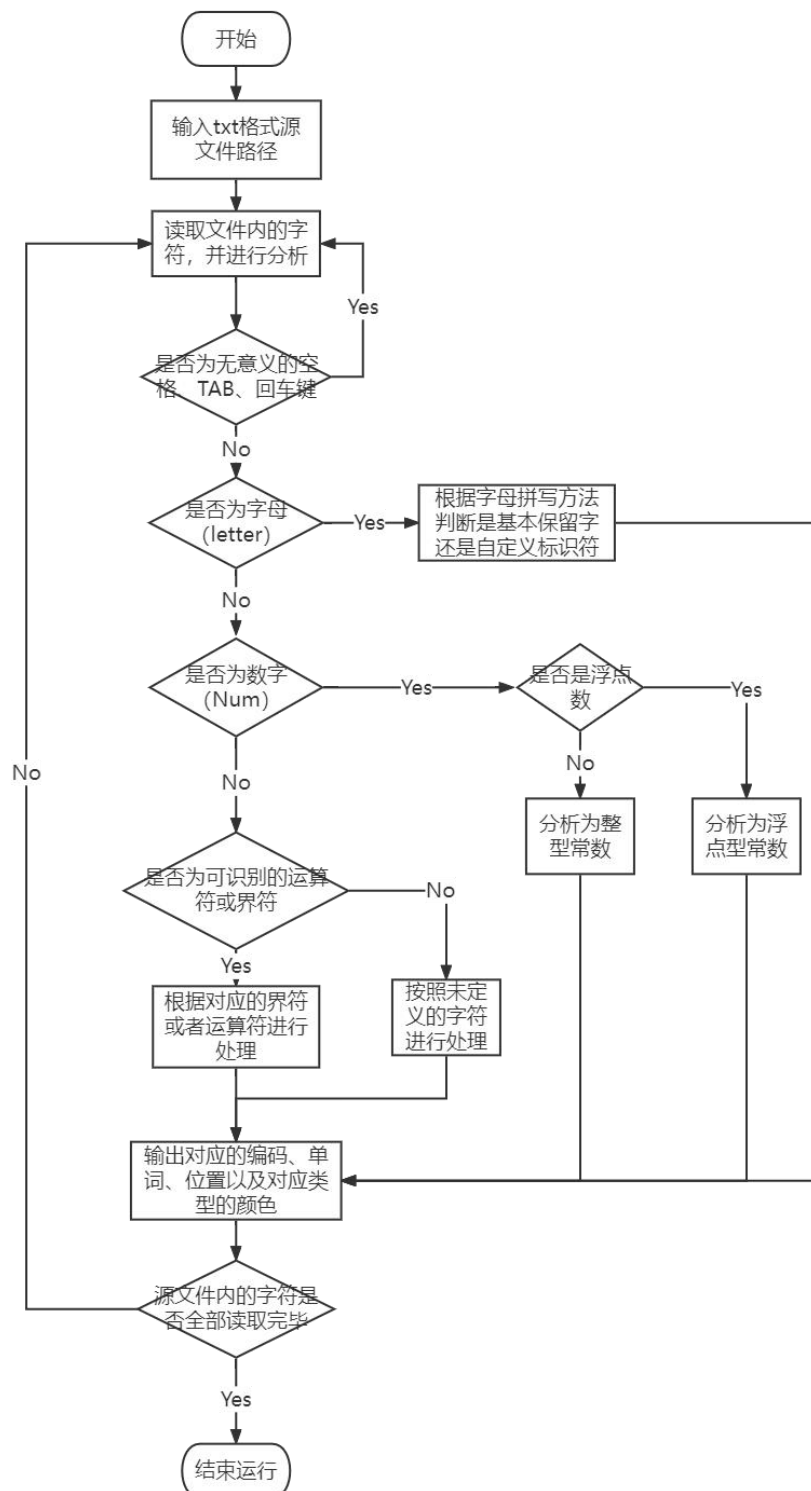


图 4.1 程序流程图

程序使用的主要函数

`isKey(string token)` //判断是否为基本保留字

//通过获取 `token` 内的单词，来首先判断是否能在基本保留字的数组中找到

//进而判断该单词是否为基本保留字

`isLetter(char letter)` //判断是否为字母

//通过获取文件内的字符，来判断是否为 `a~z` 或 `A~Z` 中的字符

//进而能够判断是否为字母，进行下一步的分析

`isNum(char Num)` //判断是否为数字

//原理同上，通过判定字符是否在 `0~9` 这个区间内

//来判断是否是数字，进行下一步的分析

`getKeyID(string token)` //返回基本保留字的编码

//获取对应 `token` 中存储的基本保留字的编码，并进行返回

`CiFa_analysis(FILE * fp)` //进行词法分析

//获取所输入路径的源文件，并对其中的内容进行词法分析

`Print(TokenCode code)` //根据不同类别，输出时选择不同的颜色

//通过获取输出时对应字符的编码，利用 `switch` 语句进行判断，

//根据判断结果判断出是那种类型的字符（五大类或未定义）

//并根据不同的类型输出不同的颜色

5 编码实现

```
#include <iostream>
#include <string>
#include <Windows.h>
using namespace std;

//对单词进行编码
enum TokenCode
{
    /*未识别*/
    UN_DEFINE = 0, //编码 code 值为 0

    /* 基本保留字 */
    KEY_VOID,      //void 关键字, 编码为 1
    KEY_MAIN,      //main 关键字, 编码为 2
    KEY_INT,        //int 关键字, 编码为 3
    KEY_DOUBLE,     //double 关键字, 编码为 4
    KEY_FOR,        //for 关键字, 编码为 5
    KEY_WHILE,      //while 关键字, 编码为 6
    KEY_SWITCH,     //switch 关键字, 编码为 7
    KEY_CASE,       //case 关键字, 编码为 8
    KEY_IF,         //if 关键字, 编码为 9
    KEY_ELSE,       //else 关键字, 编码为 10
    KEY_RETURN,     //return 关键字, 编码为 11

    /* 运算符 */
    COMPUTED_PLUS,  //+加号, 编码为 12
    COMPUTED_MINUS, //-减号, 编码为 13
    COMPUTED_STAR,  //*乘号, 编码为 14
    COMPUTED_DIVIDE, ///除号, 编码为 15
    COMPUTED_FUZH,  //:=赋值运算符, 编码为 16
    COMPUTED_EQUAL, //==等于号, 编码为 17
    COMPUTED_XIAOYU, //<小于号, 编码为 18
    COMPUTED_XIAOYUDENGYU, //<=小于等于号, 编码为 19
    COMPUTED_DAYU,  //>大于号, 编码为 20
    COMPUTED_DAYUDENGYU, //>=大于等于号, 编码为 21

    /* 界符 */
    SP_LEFT_YUAN, //(左圆括号, 编码为 22
    SP_RIGHT_YUAN, //(右圆括号, 编码为 23
    SP_LEFT_ZHONG, //[左中括号, 编码为 24
    SP_RIGHT_ZHONG, //[右中括号, 编码为 25
    SP_LEFT_BIG, //{左大括号, 编码为 26
```

```

    SP_RIGHT_BIG,    //{右大括号, 编码为 27
    SP_DOUHAO,      //{逗号, 编码为 28
    SP_FENHAO,      //{分号, 编码为 29

    /* 常数 */
    CHANG_INT,       //{整型常量, 编码为 30
    CHANG_DOUBLE,    //{浮点型常量, 编码为 31

    /* 定义标识符 */
    BIAOSHI//编码为 32
};

TokenCode code = UN_DEFINE;    //{记录单词的种别码
const int MAX = 11;            //{关键字数量
int row = 1;                    //{记录字符所在的行数
string token = "";             //{用于存储单词
char keyWord[][10] =
{ "void", "main", "int", "double", "for", "while", "switch", "case", "if", "else", "return" };
    //{关键词类型存入数组中

void print(TokenCode code)
{

    switch (code)
    {
        /*未识别的符号*/
        case UN_DEFINE:
            SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_RED);    //{
未识别的符号为红色
            cout << "( 编码为: " << code << ", 无法正确识别的符号: " << token << " , 所
在行数为: " << row << ") " << endl;
            return;
            break;
            /*基本保留字*/
        case KEY_VOID:          //{void 关键字
        case KEY_MAIN:          //{main 关键字
        case KEY_INT:           //{int 关键字
        case KEY_DOUBLE:        //{double 关键字
        case KEY_FOR:           //{for 关键字
        case KEY_WHILE:         //{while 关键字
        case KEY_SWITCH:        //{switch 关键字
        case KEY_CASE:          //{case 关键字

```

```

    case KEY_IF:      //if 关键字
    case KEY_ELSE:    //else 关键字
    case KEY_RETURN: //return 关键字
        SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_INTENSITY |
FOREGROUND_BLUE); //关键字为蓝色
        break;

        /* 运算符 */
    case COMPUTED_PLUS: //+加号
    case COMPUTED_MINUS: //-减号
    case COMPUTED_STAR: //*乘号
    case COMPUTED_DIVIDE: //除号
    case COMPUTED_FUZH: //赋值运算符
    case COMPUTED_EQUAL: //==等于号
    case COMPUTED_XIAOYU: //<小于号
    case COMPUTED_XIAOYUDENGYU: //<=小于等于号
    case COMPUTED_DAYU: //>大于号
    case COMPUTED_DAYUDENGYU: //>=大于等于号
        SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_INTENSITY |
FOREGROUND_GREEN | FOREGROUND_BLUE); //运算符为青色
        break;

        /* 界符 */
    case SP_LEFT_YUAN: //(左圆括号
    case SP_RIGHT_YUAN: //(右圆括号
    case SP_LEFT_ZHONG: //[左中括号
    case SP_RIGHT_ZHONG: //]右中括号
    case SP_LEFT_BIG: //{左大括号
    case SP_RIGHT_BIG: //{右大括号
    case SP_DOUHAO: //逗号
    case SP_FENHAO: //分号
        SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_INTENSITY |
FOREGROUND_GREEN); //分隔符为绿色
        break;

        /* 常数 */
    case CHANG_INT: //整型常量
    case CHANG_DOUBLE: //浮点型常量
        SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_INTENSITY |
FOREGROUND_RED | FOREGROUND_GREEN); //常量为黄色
        if (token.find('.') == token.npos)
            cout << "( 编码为: " << code << ", 内容为: " << atoi(token.c_str()) << " ,
所在行数为: " << row << ")" << endl; //单词为整型
        else

```

```

        cout << "( 编码为: " << code << ", 内容为:  " << atof(token.c_str()) << " ,
所在行数为: " << row << ") " << endl; //单词为浮点型
        return;
        break;
        /* 定义标识符 */
    case BIAOSHI:
        SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_INTENSITY);
        //关键字为灰色
        break;
    default:
        break;
}
    cout << "( 编码为: " << code << ", 内容为:  " << token << " ,所在行数位于: " << row
<< ") " << endl;
}
//判断是否为关键字类型

```

```

bool isKey(string token)
{
    for (int i = 0; i < MAX; i++)
    {
        if (token.compare(keyWord[i]) == 0)
            return true;
    }
    return false;
}

```

//返回基本保留字的编码

```

int getKeyID(string token)
{
    for (int i = 0; i < MAX; i++)
    {
        if (token.compare(keyWord[i]) == 0)
            return i + 1; //由于码值为0时，对应的是未定义的字符，所以在返回关键字的编码
            时，需要+1
    }
    return -1;
}

```

//判断一个字符是否是字母

```

bool isLetter(char letter)

```

```

{
    if ((letter >= 'a' && letter <= 'z') || (letter >= 'A' && letter <= 'Z'))
        return true;
    return false;
}

```

//判断一个字符是否是数字

```

bool isNum(char Num)
{
    if (Num >= '0' && Num <= '9')
        return true;
    return false;
}

```

//进行词法分析

```

void CiFa_analysis(FILE* fp)
{
    char ch;
    while ((ch = fgetc(fp)) != EOF)    //读取文件内的全部字符
    {
        token = ch;                    //将获取的字符存入 token 中
        if (ch == ' ' || ch == '\t' || ch == '\n')    //空格、Tab 和回车忽略不进行读取
        {
            if (ch == '\n')            //遇到换行符，记录行数的 row 加
1
                row++;
            continue;                //继续执行循环
        }
        else if (isLetter(ch))        //获取到以字母为开头的字符，判断是关键字还是标识
符
        {
            token = "";
            while (isLetter(ch) || isNum(ch))    //当读取到字符
            {
                token.push_back(ch);    //将读取的字符 ch 存入 token 中
                ch = fgetc(fp);        //获取下一个字符
            }
            //文件指针后退一个字节，即重新读取上述单词后的第一个字符
            fseek(fp, -1L, SEEK_CUR);
            if (isKey(token)) //关键字
                code = TokenCode(getKeyID(token));
            else //标识符

```

```

        code = BIAOSHI;    //单词为标识符
    }
    else if (isNum(ch))    //无符号常数以数字开头
    {
        int isdouble = 0; //判断是否为浮点数
        token = "";
        while (isNum(ch))
        {
            token.push_back(ch);
            ch = fgetc(fp);          //从文件中获取下一个字符
            //该单词中第一次出现小数点
            if (ch == '.' && isdouble == 0)
            {
                //小数点下一位是数字
                if (isNum(fgetc(fp)))
                {
                    isdouble = 1;    //标记该常数中已经出现过小数点
                    fseek(fp, -1L, SEEK_CUR);    //将超前读取的小数点后一位重
新读取

                    token.push_back(ch);
                    ch = fgetc(fp);          //读取小数点后的下一位数字
                }
            }
        }
        if (isdouble == 1)
            code = CHANG_DOUBLE;    //单词为浮点型
        else
            code = CHANG_INT;        //单词为整型
        //文件指针后退一个字节，即重新读取常数后的第一个字符
        fseek(fp, -1L, SEEK_CUR);
    }
    else switch (ch)
    {
        /*运算符*/
        case '+': code = COMPUTED_PLUS;    //+加号
            break;
        case '-': code = COMPUTED_MINUS;    //-减号
            break;
        case '*': code = COMPUTED_STAR;    //*乘号
            break;
        case '/': code = COMPUTED_DIVIDE;    //除号
            break;
        case '=':
    {

```

```

ch = fgetc(fp);           //超前读取'='后面的字符
if (ch == '=')           //==等于号
{
    token.push_back(ch); //将'='后面的'='存入 token 中
    code = COMPUTED_EQUAL; //单词为"=="
}
else {                   //+=赋值运算符
    code = COMPUTED_FUZH; //单词为"="
    fseek(fp, -1L, SEEK_CUR); //将超前读取的字符重新读取
}
}
break;
case '<':
{
    ch = fgetc(fp);           //超前读取'<'后面的字符
    if (ch == '=')           //<=小于等于号
    {
        token.push_back(ch); //将'<'后面的'='存入 token 中
        code = COMPUTED_XIAOYUDENGYU; //单词为"<="
    }
    else {                 //<小于号
        code = COMPUTED_XIAOYU; //单词为"<"
        fseek(fp, -1L, SEEK_CUR); //将超前读取的字符重新读取
    }
}
break;
case '>':
{
    ch = fgetc(fp);           //超前读取'>'后面的字符
    if (ch == '=')           //>=大于等于号
    {
        token.push_back(ch); //将'>'后面的'='存入 token 中
        code = COMPUTED_DAYUDENGYU; //单词为">="
    }
    else {                 //>大于号
        code = COMPUTED_DAYU; //单词为">"
        fseek(fp, -1L, SEEK_CUR); //将超前读取的字符重新读取
    }
}
break;
/*界符*/
case '(': code = SP_LEFT_YUAN; // (左圆括号
break;
case ')': code = SP_RIGHT_YUAN; // )右圆括号

```



```

        break;
        case '[': code = SP_LEFT_ZHONG;           //[左中括号
            break;
        case ']': code = SP_RIGHT_ZHONG;         //[右中括号
            break;
        case '{': code = SP_LEFT_BIG;            //{左大括号
            break;
        case '}': code = SP_RIGHT_BIG;           //{右大括号
            break;
        case ',': code = SP_DOUHAO;              //,逗号
            break;
        case ';': code = SP_FENHAO;              //;分号
            break;
        //未识别内容
        default: code = UN_DEFINE;
    }
    print(code);                                //打印词法分析结果
}

}

int main()
{
    string filename;                            //文件路径
    FILE* fp;                                   //文件指针
    cout << "请输入所要进行词法分析的 txt 文件的路径: " << endl;
    while (true) {
        cin >> filename;                        //读取文件路径
        if ((fopen_s(&fp, filename.c_str(), "r")) == 0) //打开文件
            break;
        else
            cout << "路径有误, 请重新输入: " << endl; //读取失败
    }
    cout << "词法分析正在加载中....." <<
endl;
    cout << "....." <<
endl;
    cout << "....." <<
endl;
    cout << "....." <<
endl;
    cout << "....." <<
endl;
    cout << "....." <<
endl;

```

```

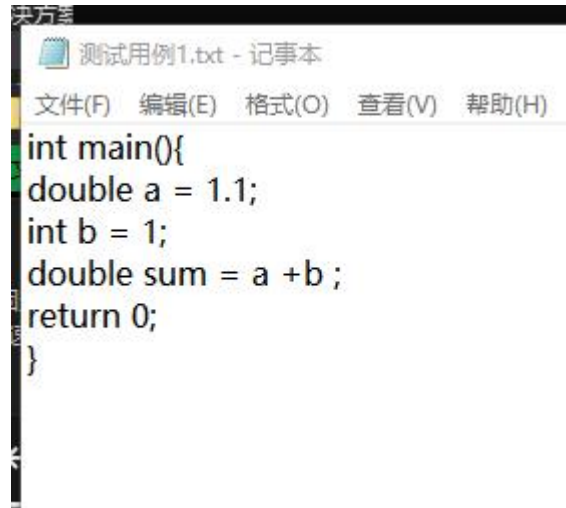
        cout << "-----" <<
endl;
        cout << "-----" <<
endl;
        cout << "-----词法分析结果如下：" <<
endl;
        cout << "未识别的符号是红色；" << endl;
        cout << "基本保留字类型是蓝色；" << endl;
        cout << "运算符类型是青色；" << endl;
        cout << "界符类型是绿色；" << endl;
        cout << "常数类型是黄色；" << endl;
        cout << "定义的标识符是灰色；" << endl << endl;
        CiFa_analysis(fp);           //词法分析
        fclose(fp);                 //关闭文件
        cout << endl<<endl<<endl<<endl;
        cout << "-----阅览说明-----" << endl;
        cout << "编码为 0 时，字符是未定义的" << endl;
        cout << "编码为 1 时，字符是 void" << endl;
        cout << "编码为 2 时，字符是 main" << endl;
        cout << "编码为 3 时，字符是 int" << endl;
        cout << "编码为 4 时，字符是 double" << endl;
        cout << "编码为 5 时，字符是 for" << endl;
        cout << "编码为 6 时，字符是 while" << endl;
        cout << "编码为 7 时，字符是 switch" << endl;
        cout << "编码为 8 时，字符是 case" << endl;
        cout << "编码为 9 时，字符是 if" << endl;
        cout << "编码为 10 时，字符是 else" << endl;
        cout << "编码为 11 时，字符是 return" << endl;
        cout << "编码为 12 时，字符是 +" << endl;
        cout << "编码为 13 时，字符是 -" << endl;
        cout << "编码为 14 时，字符是 *" << endl;
        cout << "编码为 15 时，字符是 /" << endl;
        cout << "编码为 16 时，字符是 =" << endl;
        cout << "编码为 17 时，字符是 ==" << endl;
        cout << "编码为 18 时，字符是 <" << endl;
        cout << "编码为 19 时，字符是 <=" << endl;
        cout << "编码为 20 时，字符是 >" << endl;
        cout << "编码为 21 时，字符是 >=" << endl;
        cout << "编码为 22 时，字符是 (" << endl;
        cout << "编码为 23 时，字符是 )" << endl;
        cout << "编码为 24 时，字符是 [" << endl;
        cout << "编码为 25 时，字符是 "]" << endl;
        cout << "编码为 26 时，字符是 {" << endl;
        cout << "编码为 27 时，字符是 }" << endl;

```

```
cout << "编码为 28 时, 字符是  ," << endl;
cout << "编码为 29 时, 字符是  ;" << endl;
cout << "编码为 30 时, 字符是  int" << endl;
cout << "编码为 31 时, 字符是  double" << endl;
cout << "编码为 32 时, 字符是自定义标识符" << endl<<endl<<endl;
cout << "请根据以上说明继续阅览词法分析结果" << endl;
system("pause");
return 0;
}
```

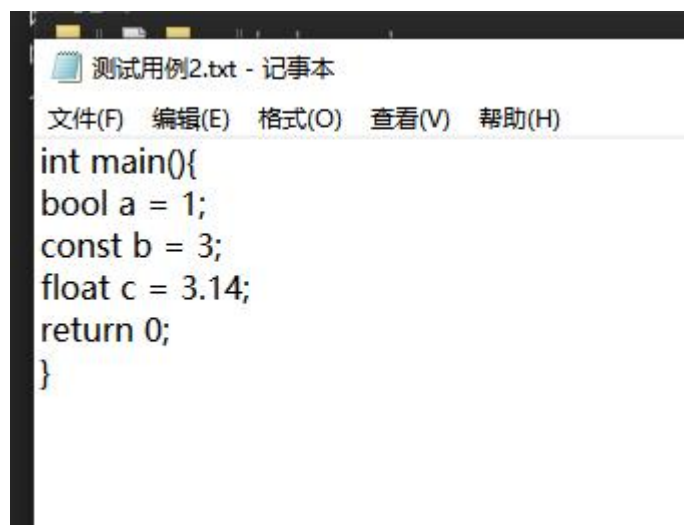
6 测试和试运行

主要是采用的黑盒测试，我主要构建了三个测试用例分别来进行测试，三个测试用例文件的内容分别如下图所示：



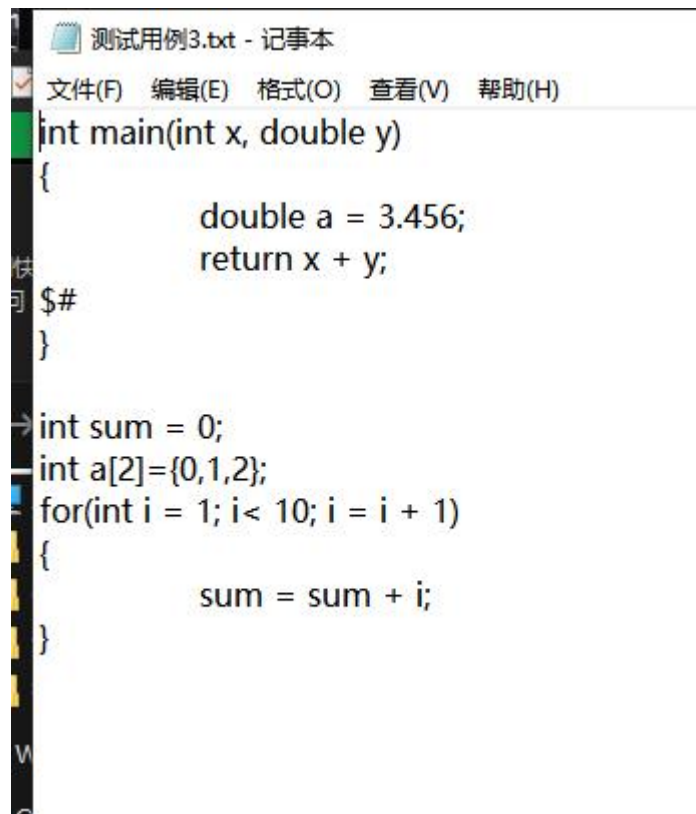
```
int main(){  
double a = 1.1;  
int b = 1;  
double sum = a + b ;  
return 0;  
}
```

图 6.1 测试用例 1



```
int main(){  
bool a = 1;  
const b = 3;  
float c = 3.14;  
return 0;  
}
```

图 6.2 测试用例 2



```
测试用例3.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

int main(int x, double y)
{
    double a = 3.456;
    return x + y;
}

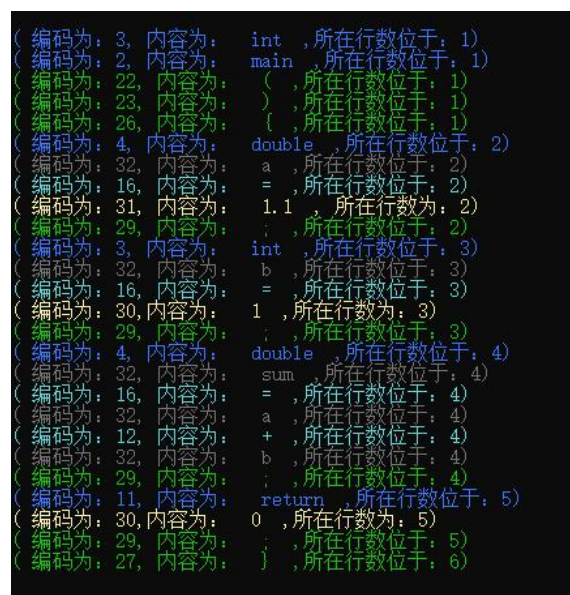
int sum = 0;
int a[2]={0,1,2};
for(int i = 1; i < 10; i = i + 1)
{
    sum = sum + i;
}
```

图 6.3 测试用例 3

最终测试结果如下：

(1) 测试结果 1:

能够正确分析测试用例 1 里所有的字符，并显示其对应的编码和所在位置，并根据不同类别输出不同的颜色。



```
( 3, 内容为: int ,所在行数位于: 1)
( 2, 内容为: main ,所在行数位于: 1)
( 22, 内容为: ( ,所在行数位于: 1)
( 23, 内容为: ) ,所在行数位于: 1)
( 26, 内容为: { ,所在行数位于: 1)
( 4, 内容为: double ,所在行数位于: 2)
( 32, 内容为: a ,所在行数位于: 2)
( 16, 内容为: = ,所在行数位于: 2)
( 31, 内容为: 1.1 ,所在行数位于: 2)
( 29, 内容为: ; ,所在行数位于: 2)
( 3, 内容为: int ,所在行数位于: 3)
( 32, 内容为: b ,所在行数位于: 3)
( 16, 内容为: = ,所在行数位于: 3)
( 30, 内容为: 1 ,所在行数位于: 3)
( 29, 内容为: ; ,所在行数位于: 3)
( 4, 内容为: double ,所在行数位于: 4)
( 32, 内容为: sum ,所在行数位于: 4)
( 16, 内容为: = ,所在行数位于: 4)
( 32, 内容为: a ,所在行数位于: 4)
( 12, 内容为: + ,所在行数位于: 4)
( 32, 内容为: b ,所在行数位于: 4)
( 29, 内容为: ; ,所在行数位于: 4)
( 11, 内容为: return ,所在行数位于: 5)
( 30, 内容为: 0 ,所在行数位于: 5)
( 29, 内容为: ; ,所在行数位于: 5)
( 27, 内容为: } ,所在行数位于: 6)
```

图 6.4 测试结果 1

(2) 测试结果 2:

大部分字符分析无误，但是在分析到“bool”、“const”、“float”这三个非自定义标识符时，却将其识别为自定义标识符，出现错误。

分析原因：在给基本保留字进行编码时，选择的基本保留字有限，只对 11 个常用的基本保留字进行了编码，因此在识别到未编码的基本保留字时，系统会将其视为自定义标识符。

```
( 编码为: 3, 内容为: int ,所在行数位于: 1)
( 编码为: 2, 内容为: main ,所在行数位于: 1)
( 编码为: 22, 内容为: ( ,所在行数位于: 1)
( 编码为: 23, 内容为: ) ,所在行数位于: 1)
( 编码为: 26, 内容为: { ,所在行数位于: 1)
( 编码为: 32, 内容为: bool ,所在行数位于: 2)
( 编码为: 32, 内容为: a ,所在行数位于: 2)
( 编码为: 16, 内容为: = ,所在行数位于: 2)
( 编码为: 30, 内容为: 1 ,所在行数位于: 2)
( 编码为: 29, 内容为: ; ,所在行数位于: 2)
( 编码为: 32, 内容为: const ,所在行数位于: 3)
( 编码为: 32, 内容为: b ,所在行数位于: 3)
( 编码为: 16, 内容为: = ,所在行数位于: 3)
( 编码为: 30, 内容为: 3 ,所在行数位于: 3)
( 编码为: 29, 内容为: ; ,所在行数位于: 3)
( 编码为: 32, 内容为: float ,所在行数位于: 4)
( 编码为: 32, 内容为: c ,所在行数位于: 4)
( 编码为: 16, 内容为: = ,所在行数位于: 4)
( 编码为: 31, 内容为: 3.14 ,所在行数位于: 4)
( 编码为: 29, 内容为: ; ,所在行数位于: 4)
( 编码为: 11, 内容为: return ,所在行数位于: 5)
( 编码为: 30, 内容为: 0 ,所在行数位于: 5)
( 编码为: 29, 内容为: ; ,所在行数位于: 5)
( 编码为: 27, 内容为: } ,所在行数位于: 6)
```

图 6.5 测试结果 2

(3) 测试结果 3:

字符分析无误，并且能够将未定义的字符“#”和“\$”识别出来，并且得出所在行数，用红色文字进行输出；其他字符分析无误。

```

( 编码为: 3, 内容为: int ,所在行数位于: 1)
( 编码为: 2, 内容为: main ,所在行数位于: 1)
( 编码为: 22, 内容为: ( ,所在行数位于: 1)
( 编码为: 3, 内容为: int ,所在行数位于: 1)
( 编码为: 32, 内容为: x ,所在行数位于: 1)
( 编码为: 28, 内容为: , ,所在行数位于: 1)
( 编码为: 4, 内容为: double ,所在行数位于: 1)
( 编码为: 32, 内容为: y ,所在行数位于: 1)
( 编码为: 23, 内容为: ) ,所在行数位于: 1)
( 编码为: 26, 内容为: { ,所在行数位于: 2)
( 编码为: 4, 内容为: double ,所在行数位于: 3)
( 编码为: 32, 内容为: a ,所在行数位于: 3)
( 编码为: 16, 内容为: = ,所在行数位于: 3)
( 编码为: 31, 内容为: 3.456 ,所在行数为: 3)
( 编码为: 29, 内容为: ; ,所在行数位于: 3)
( 编码为: 11, 内容为: return ,所在行数位于: 4)
( 编码为: 32, 内容为: x ,所在行数位于: 4)
( 编码为: 12, 内容为: + ,所在行数位于: 4)
( 编码为: 32, 内容为: y ,所在行数位于: 4)
( 编码为: 29, 内容为: ; ,所在行数位于: 4)
( 编码为: 0, 无法正确识别的符号: $ ,所在行数为: 5)
( 编码为: 0, 无法正确识别的符号: # ,所在行数为: 5)
( 编码为: 27, 内容为: } ,所在行数位于: 6)
( 编码为: 3, 内容为: int ,所在行数位于: 8)
( 编码为: 32, 内容为: sum ,所在行数位于: 8)
( 编码为: 16, 内容为: = ,所在行数位于: 8)
( 编码为: 30, 内容为: 0 ,所在行数为: 8)
( 编码为: 29, 内容为: ; ,所在行数位于: 8)
( 编码为: 3, 内容为: int ,所在行数位于: 9)
( 编码为: 32, 内容为: a ,所在行数位于: 9)
( 编码为: 24, 内容为: [ ,所在行数位于: 9)
( 编码为: 30, 内容为: 2 ,所在行数为: 9)
( 编码为: 25, 内容为: ] ,所在行数位于: 9)
( 编码为: 16, 内容为: = ,所在行数位于: 9)
( 编码为: 26, 内容为: { ,所在行数位于: 9)
( 编码为: 30, 内容为: 0 ,所在行数为: 9)
( 编码为: 28, 内容为: , ,所在行数位于: 9)
( 编码为: 30, 内容为: 1 ,所在行数为: 9)
( 编码为: 28, 内容为: , ,所在行数位于: 9)
( 编码为: 30, 内容为: 2 ,所在行数为: 9)
( 编码为: 27, 内容为: } ,所在行数位于: 9)
( 编码为: 29, 内容为: ; ,所在行数位于: 9)
( 编码为: 5, 内容为: for ,所在行数位于: 10)

```

图 6.6 测试结果 3 (上)


```

( 编码为: 3, 内容为: int ,所在行数为: 9)
( 编码为: 32, 内容为: a ,所在行数为: 9)
( 编码为: 24, 内容为: [ ,所在行数为: 9)
( 编码为: 30, 内容为: 2 ,所在行数为: 9)
( 编码为: 25, 内容为: ] ,所在行数为: 9)
( 编码为: 16, 内容为: = ,所在行数为: 9)
( 编码为: 26, 内容为: { ,所在行数为: 9)
( 编码为: 30, 内容为: 0 ,所在行数为: 9)
( 编码为: 28, 内容为: , ,所在行数为: 9)
( 编码为: 30, 内容为: 1 ,所在行数为: 9)
( 编码为: 28, 内容为: , ,所在行数为: 9)
( 编码为: 30, 内容为: 2 ,所在行数为: 9)
( 编码为: 27, 内容为: } ,所在行数为: 9)
( 编码为: 29, 内容为: ; ,所在行数为: 9)
( 编码为: 5, 内容为: for ,所在行数为: 10)
( 编码为: 22, 内容为: ( ,所在行数为: 10)
( 编码为: 3, 内容为: int ,所在行数为: 10)
( 编码为: 32, 内容为: i ,所在行数为: 10)
( 编码为: 16, 内容为: = ,所在行数为: 10)
( 编码为: 30, 内容为: 1 ,所在行数为: 10)
( 编码为: 29, 内容为: ; ,所在行数为: 10)
( 编码为: 32, 内容为: i ,所在行数为: 10)
( 编码为: 18, 内容为: < ,所在行数为: 10)
( 编码为: 30, 内容为: 10 ,所在行数为: 10)
( 编码为: 29, 内容为: ; ,所在行数为: 10)
( 编码为: 32, 内容为: i ,所在行数为: 10)
( 编码为: 16, 内容为: = ,所在行数为: 10)
( 编码为: 32, 内容为: i ,所在行数为: 10)
( 编码为: 12, 内容为: + ,所在行数为: 10)
( 编码为: 30, 内容为: 1 ,所在行数为: 10)
( 编码为: 23, 内容为: ) ,所在行数为: 10)
( 编码为: 26, 内容为: { ,所在行数为: 11)
( 编码为: 32, 内容为: sum ,所在行数为: 12)
( 编码为: 16, 内容为: = ,所在行数为: 12)
( 编码为: 32, 内容为: sum ,所在行数为: 12)
( 编码为: 12, 内容为: + ,所在行数为: 12)
( 编码为: 32, 内容为: i ,所在行数为: 12)
( 编码为: 29, 内容为: ; ,所在行数为: 12)
( 编码为: 27, 内容为: } ,所在行数为: 13)

```

图 6.7 测试结果 3（下）

（4）测试结果 4（测试可视化部分）：

开头的说明部分和结尾的说明部分都显示无误。

```

请输入所要进行词法分析的txt文件的路径:
D:\桌面小工具\临时文件夹3\大三\编译原理\课设\keshe_wendang\测试用例3.txt
词法分析正在加载中.....

-----词法分析结果如下:

未识别的符号是红色;
基本保留字类型是蓝色;
运算符类型是青色;
界符类型是绿色;
常数类型是黄色;
定义的标识符是灰色;
( 编码为: 3, 内容为: int ,所在行数为: 1)

```

图 6.8 测试结果 4（上）


```

-----浏览说明-----
编码为0时, 字符是未定义的
编码为1时, 字符是 void
编码为2时, 字符是 main
编码为3时, 字符是 int
编码为4时, 字符是 double
编码为5时, 字符是 for
编码为6时, 字符是 while
编码为7时, 字符是 switch
编码为8时, 字符是 case
编码为9时, 字符是 if
编码为10时, 字符是 else
编码为11时, 字符是 return
编码为12时, 字符是 +
编码为13时, 字符是 -
编码为14时, 字符是 *
编码为15时, 字符是 /
编码为16时, 字符是 =
编码为17时, 字符是 ==
编码为18时, 字符是 <
编码为19时, 字符是 <=
编码为20时, 字符是 >
编码为21时, 字符是 >=
编码为22时, 字符是 (
编码为23时, 字符是 )
编码为24时, 字符是 [
编码为25时, 字符是 ]
编码为26时, 字符是 {
编码为27时, 字符是 }
编码为28时, 字符是 ,
编码为29时, 字符是 :
编码为30时, 字符是 int
编码为31时, 字符是 double
编码为32时, 字符是自定义标识符

```

请根据以上说明继续浏览词法分析结果

D:\c++daima\bianyiyuanli keshe\Debug\bianyiyuanli keshe

图 6.9 测试结果 4（下）

7. 总结

本次课程设计，我利用 C++ 简单实现了从输入的源程序中，识别出各个具有独立意义的单词，能够简单实现编译原理中词法分析的部分；但是，我需要改进的地方还有很多，比如基本保留字的类型可能要比我代码里罗列的要更多（我只对 11 个常用的基本保留字进行了分析识别），在容纳量和分析量方面还有很大的改进空间；我的交互界面设计的也不太尽如人意，只是有着简单的说明和分析结果的展示，在以后我也可以将这一词法分析程序的可视化部分做更多的改进，让用户使用起来更加容易上手，视觉体验也更棒；在后期和老师的交流中，也发现了自己的项目中仍存在一些自己认为已经解决但实际上并没有真正解决的问题（比如常数输出时输出的是字符串而不是对应的常数这一情况），我也会在今后的项目经历中，对自己解决的问题多思考多研究，也主动和老师沟通，这样才能让我获得更多不同思考角度的答案，也收获更多的知识。

通过此次课程设计，我进一步了解了什么是词法分析，熟悉了词法分析的状态转换图，对课本上的知识有了更深的理解。我发现课程设计对动手实践的能力要求比较严格，都需要亲自操作才能够顺利地完成自己的实验，我们要真正地去模拟这样的操作流程，才能更好地掌握专业知识，而我也坚信，这些都必定会对我们以后的生活方式产生重大的影响。在这个过程中，我深深体会到了学习带来的快乐与满足感。

8. 参考文献

- [1] 王生原 董渊 张素琴 吕映芝 蒋维杜.《编译原理（第 3 版）》.北京:清华大学出版社, 2015-6
- [2] <https://blog.csdn.net/ministarler/article/details/12176697>
- [3] 朱朝霞.编译词法分析程序实现探讨.《长春工程学院学报：自然科学版》,2011,4:132-134
- [4] https://blog.csdn.net/qq_41985293/article/details/106290290/