代码实现的功能

针对文件中的表达式，能够提取 token，生成语法树，逆波兰式以及四元式，生成汇编指令。

样例运行结果如下：

```
划分 token 如下：
*****************************************
<ID letter>
<PLUS +>
<NUM 23>
<TIMES *>
<LPAREN (>
<NUM 5>
<MINUS ->
<ID b>
<RPAREN )>
<OVER />
<NUM 7>

*********************************************
成功建立语法树！*****************************
当前树结构如下*********************************
        7
    /
                b
            -
                5
        *
            23
+
    letter

生成逆波兰式如下*******************************
letter 23 5 b - * 7 / +
生成四元式如下*******************************
- 5,b,PH
汇编指令如下:
        mov ax 5
        add ax b
        mov PH 5
* 23,PH,QG
汇编指令如下:
```

```
        mov ax 23
        add ax PH
        mov QG 23
/ QG,7,HU
汇编指令如下:
        mov ax QG
        add ax 7
        mov HU QG
+ letter,HU,ME
汇编指令如下:
        mov ax letter
        add ax HU
        mov ME letter
```

实现思路说明：

1.逐个字符读取文件文件中的表达式, 划分为 NUM, ID, +, -, *, /几个类型, 存储<type,value>格式于结构体数组中。
2.针对如下文法，采用递归下降方法生成语法树

exp->term+term||term-term

term->factor*factor||term->factor/factor

factor->NUM||ID||(exp)

3.后序遍历语法树，输出逆波兰式，并存储到 vector 中
4.采用栈结构对第三步获得的逆波兰式处理生成四元式
5. 对第四步获得的四元式处理获得汇编指令。

数据结构说明

```
typedef enum          //表达式中 token 类型
{
    ID,NUM,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,END
} TokenType;

typedef struct scanToken     //token 结构
{
    TokenType op;
    char stringValue[256];
} TokenNode;
```

```
typedef struct treeNode     //语法树节点结构
{
    struct treeNode * child[MAXCHILDREN];
    TokenNode attr;
} TreeNode;

TokenNode token;    //单个 token，用于构造语法树，是 expr 的元素
TokenNode expr[256]; //用一个结构体数组存放识别出的 token,类似<attr,value>
int   flag=0;      //用于指示 expr 下标
stack<TokenNode> OPND;   //存放操作数
vector<TokenNode> rpn;    //存放得到的逆波兰式，以便后续操作
TokenNode Vtemp[10]={       //生成四元式时存放中间结果

{ID,"t0"},{ID,"t1"},{ID,"t2"},{ID,"t3"},{ID,"t4"},{ID,"t5"},{ID,"t6"},{ID,"t7"},{ID,"t8"},{ID,"t9"}

};
```

函数模块说明

```
void scan(TokenNode* tokenlist,char* filename);      //扫描获得 token，不支持多字符标识符和数字
void fscan(TokenNode* expression,char* filename);    ////扫描文件获得 token,支持多字符标识符和数字
void printToken(TokenNode* expression);   //输出已识别的 token
TreeNode * exp();    //exp->term+term||term-term
TreeNode * term();   //term->factor*factor||term->factor/factor
TreeNode * factor();//factor->NUM||ID||(exp)
TreeNode * newNode(); //构造语法树的过程中分配树节点
void TreePrint(TreeNode * T,int level);   //按照树形打印
void postOrderTraverse(TreeNode *T); //后序遍历语法树
void printFef(vector<TokenNode> fexp); //输出四元式及汇编指令
char *rand_str(char *str); //随机字符串生成
```

出错处理

在词法分析中，遇到无法识别的符号，会提示：符号无法识别,请检查源文件！，并跳过该符号继续分析。
在语法分析中，如果不符合语法规则，会提示：unexpected token;,程序也会终止。
分配树节点内存时会有内存不足的错误提示。

代码优点

函数命名较清晰，大致上是按照整个编译器的处理过程在编写。输出格式较规范。

不足及改进之处

1. 将词法分析的结果存入定常结构体数组，再进行语法分析，导致无法支持过长的表达式，而且比较浪费内存。后续会写 getToken 函数直接传单个 token 给语法分析过程，不再做中间存储。
2. 尝试过多文件，但在链接时会出现变量重定义的情况，不太好改，可能是因为指针用的不熟，只能采用全局变量。正在改进。
3. 有一个过长函数。因为是词法分析过程，考虑分支较多，代码可能很长，后续如果采用状态机可能会更长，暂不知如何改进。

测试

构造了几个表达式，把所有函数中出现的分支基本都执行了一遍，针对出错情况也有测试。请将测试用例文件名改为 exp1，再进行测试。