

编译原理

**实验名称**

学 院： 詹天佑学院

专 业： 计算机科学与技术

学生姓名： 程维森

学 号： 21231264

北京交通大学

2023年11月

1. **程序功能描述**

注意：  
 递归下降分析对文法有特定要求：文法必须符合LL(1)文法的条件。LL(1)文法的要求包括两点：首先，文法不能包含左递归；其次，文法不能引起回溯。如果文法存在左递归或者回溯的情况，就需要在进行递归下降分析之前对文法进行修改，确保它不包含这两种情况。

这个程序是一个递归下降的语法分析器，用于判断输入的字符串是否符合特定的文法规则。它实现了一个简单的算术表达式语言的语法分析。以下是程序的主要功能和流程：

1. 文件读取：程序从名为`test.txt`的文件中读取一行字符串，将该字符串存储在变量`str`中。

2. 字符串处理：将读取到的字符串中的字母替换为字符`'i'`，将所有变量标识符标准化为`'i'`。处理后的字符串存储在变量`s`中。

3. 语法分析：程序通过一系列的递归下降函数（例如`checkS()`, `checkE()`, `checkT()`等）来检查字符串`s`是否符合特定的文法规则。每个函数负责检查文法的一个部分，如果符合规则则继续递归分析下一个部分，如果不符合规则则输出错误信息，并将`flag`置为1，表示出现错误。具体的文法规则在代码中有详细的注释说明。

4. 错误处理：如果在分析过程中遇到不符合规则的部分，将`flag`置为1，同时输出错误信息，指示出现错误的位置（`amt`变量记录当前分析的位置）。

5. 文件输出：将分析结果输出到名为`output.txt`的文件中。如果字符串符合文法规则，输出"legal"，否则输出"illegal"。

6. 程序结束：输出"program complete!"表示程序运行结束。

对递归下降的详细分析：（详细的分析请参考后续框图）

* 分析S规则：

checkS函数首先检查是否是以字母'i'开头，然后检查是否有等号'='，之后继续分析表达式部分。

如果不符合规则，将flag置为1，并输出错误信息。

* 分析E规则：

checkE函数检查表达式部分，它调用了checkT函数，再调用checkE\_函数。

checkT函数检查是否是以左括号'('或字母'i'开头，如果是，调用checkF函数，再根据后续字符决定继续递归或返回。

checkF函数检查因子，可以是括号内的表达式或者单个变量，然后递归处理。

checkE\_函数检查是否是加号'+'或减号'-'，如果是，调用checkADD函数，然后继续递归。

* 分析T规则：

checkT函数类似地检查项，包括因子部分的递归处理，乘法和除法的检查，以及递归调用checkT\_函数。

* 分析checkT\_函数：

checkT\_函数用于处理T规则中的乘法和除法操作。在递归下降分析中，T规则可以表示为T -> FT'，其中T'表示可能存在的乘法或除法操作。

checkT\_函数首先检查当前字符（s[amt]）是否为乘法（'\*'）或除法（'/'）操作符。

如果是乘法或除法操作符，函数调用checkMULT()进行操作符的匹配，然后递归调用checkF()来处理因子（F）部分。

接着，checkT\_函数递归调用自身，继续检查下一个可能的乘法或除法操作。

如果当前字符不是乘法或除法操作符，或者因子部分不符合规则，函数会返回，使得控制流回到调用它的地方。如果在递归调用的过程中发现错误，flag会被设置为1，表示出现了语法错误。

* 分析checkE\_函数：

checkE\_函数用于处理E规则中的加法和减法操作。在递归下降分析中，E规则可以表示为E -> TE'，其中E'表示可能存在的加法或减法操作。

checkE\_函数首先检查当前字符（s[amt]）是否为加法（'+'）或减法（'-'）操作符。

如果是加法或减法操作符，函数调用checkADD()进行操作符的匹配，然后递归调用checkT()来处理项（T）部分。

接着，checkE\_函数递归调用自身，继续检查下一个可能的加法或减法操作。

1. **主要数据结构**

|  |  |
| --- | --- |
| 变量及类型 | 用途 |
| ifstream input | 文件输入流，从文件中读取输入数据 |
| ofstream output | 文件输出流，将输出结果写入文件 |
| string str, s | 存储从文件中读取的字符串和处理后的字符串 |
| int flag | 标志变量，用于标记语法分析过程中的错误 |
| int amt | 计数器，用于跟踪分析的字符串位置 |
| void checkF() | 检查语法中的非终结符F |
| void checkS() | 检查语法中的非终结符S |
| void checkMUlT() | 检查乘法操作符（\*和/） |
| void checkT\_() | 辅助函数，用于递归下降分析中的T\_部分 |
| void checkT() | 检查语法中的非终结符T |
| void checkVOC() | 检查语法中的非终结符V（变量） |
| void checkADD() | 检查加法操作符（+和-） |
| void checkE() | 检查语法中的非终结符E |
| void checkE\_() | 辅助函数，用于递归下降分析中的E\_部分 |
| void fcheck() | 检查特殊字符#，用于程序的终止 |
| int main() | 主函数，控制程序的流程和执行逻辑 |

1. **程序结构描述**
2. **设计方法**

本程序采用了递归下降分析方法来实现语法分析。递归下降分析是一种自顶向下的语法分析方法，它从顶层文法符号开始，递归地分解输入，直到达到最底层的终结符或者错误。以下是程序的设计方法分析：

主要设计方法：

1. 递归下降分析：程序使用了一系列的递归下降函数（`checkS()`, `checkE()`, `checkT()`, 等）来逐步检查输入字符串的各个部分是否符合特定的文法规则。每个函数负责检查文法的一个部分，如果符合规则则继续递归分析下一个部分，如果不符合规则则输出错误信息，并将`flag`置为1，表示出现错误。

2. 错误处理：当发现语法错误时，程序会输出错误信息，指明出错的位置（`ERROR num is`）并将`flag`置为1。这样，在发现第一个错误后，程序会停止继续分析，确保在错误发生后不会继续输出后续的错误信息。

3. 输入处理： 输入的字符串被处理成符合文法规则的形式。例如，字母字符会被替换成文法中的终结符"i"，并在字符串的末尾添加结束符"#”。

4. 输出：程序通过输出流将分析过程和结果输出到文件中（"output.txt"），同时也在控制台输出提示信息，如"program complete!"。

程序执行流程：

1. 读取输入文件内容并进行预处理，将字母字符替换为文法符号"i"，在字符串末尾添加结束符"#"。

2. 调用`checkS()`函数开始递归下降分析。

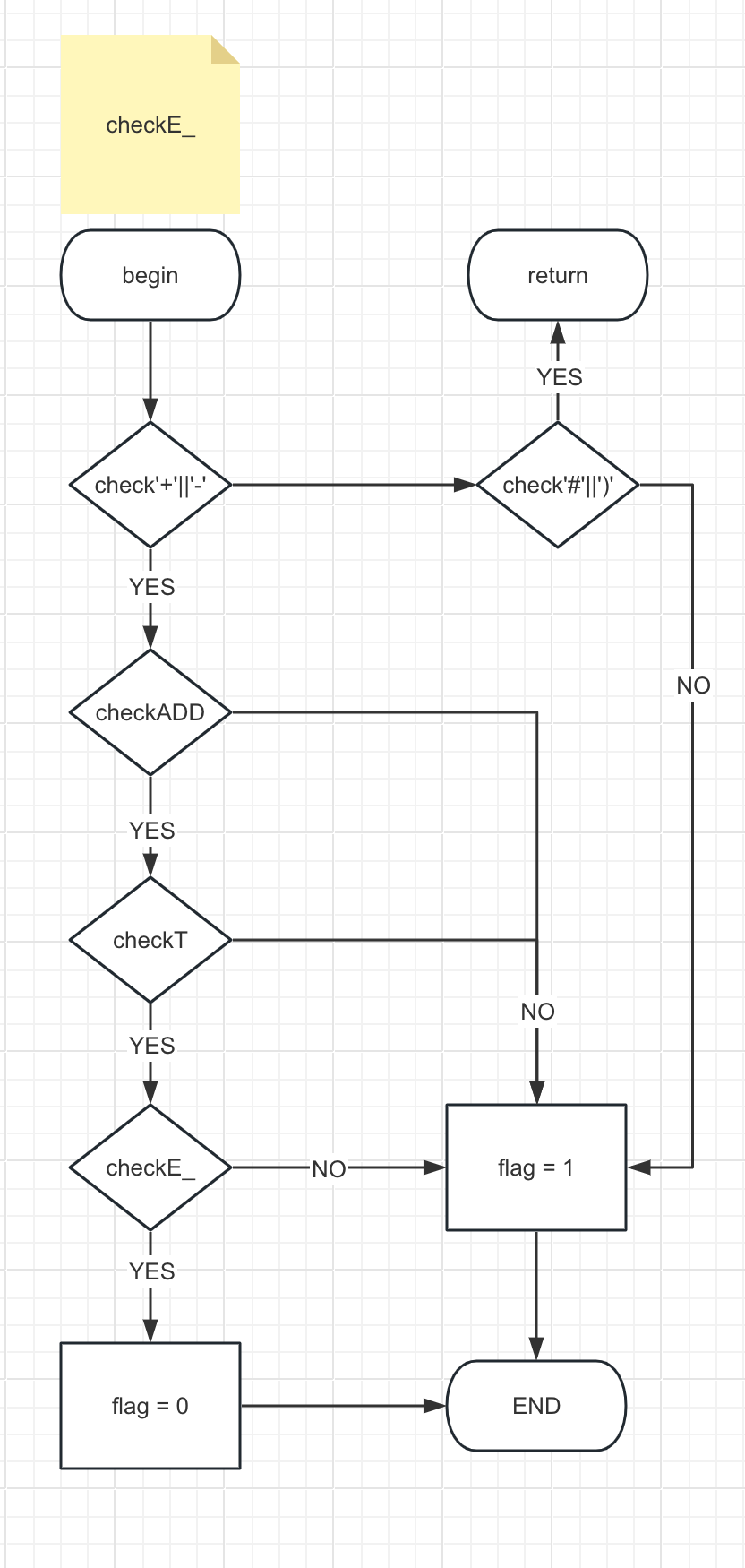
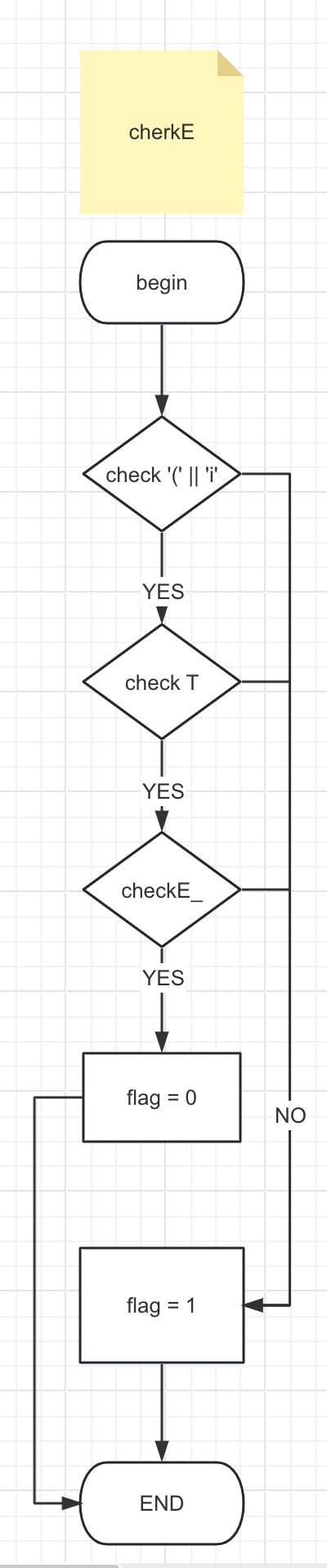
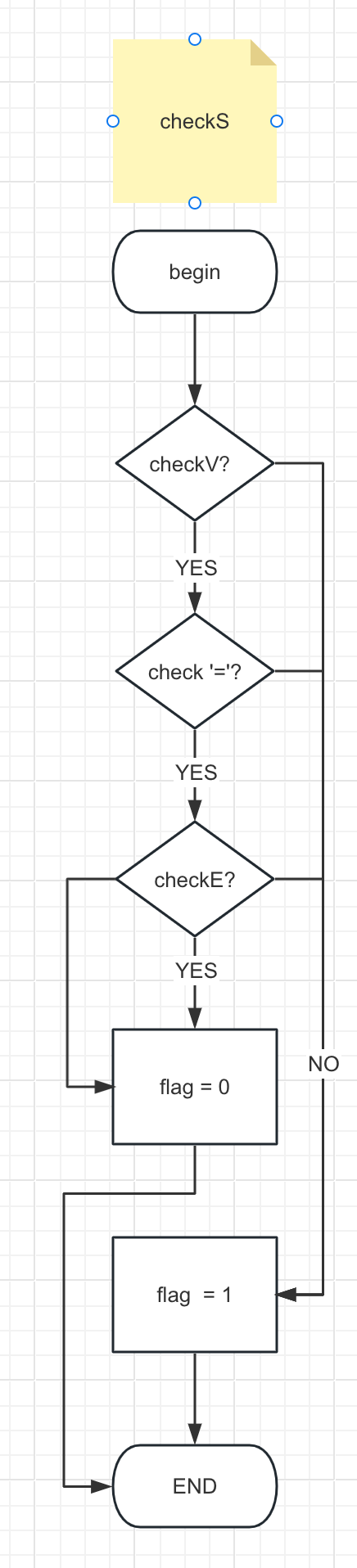
3. 递归下降分析的各个函数按照文法规则逐步检查输入字符串。

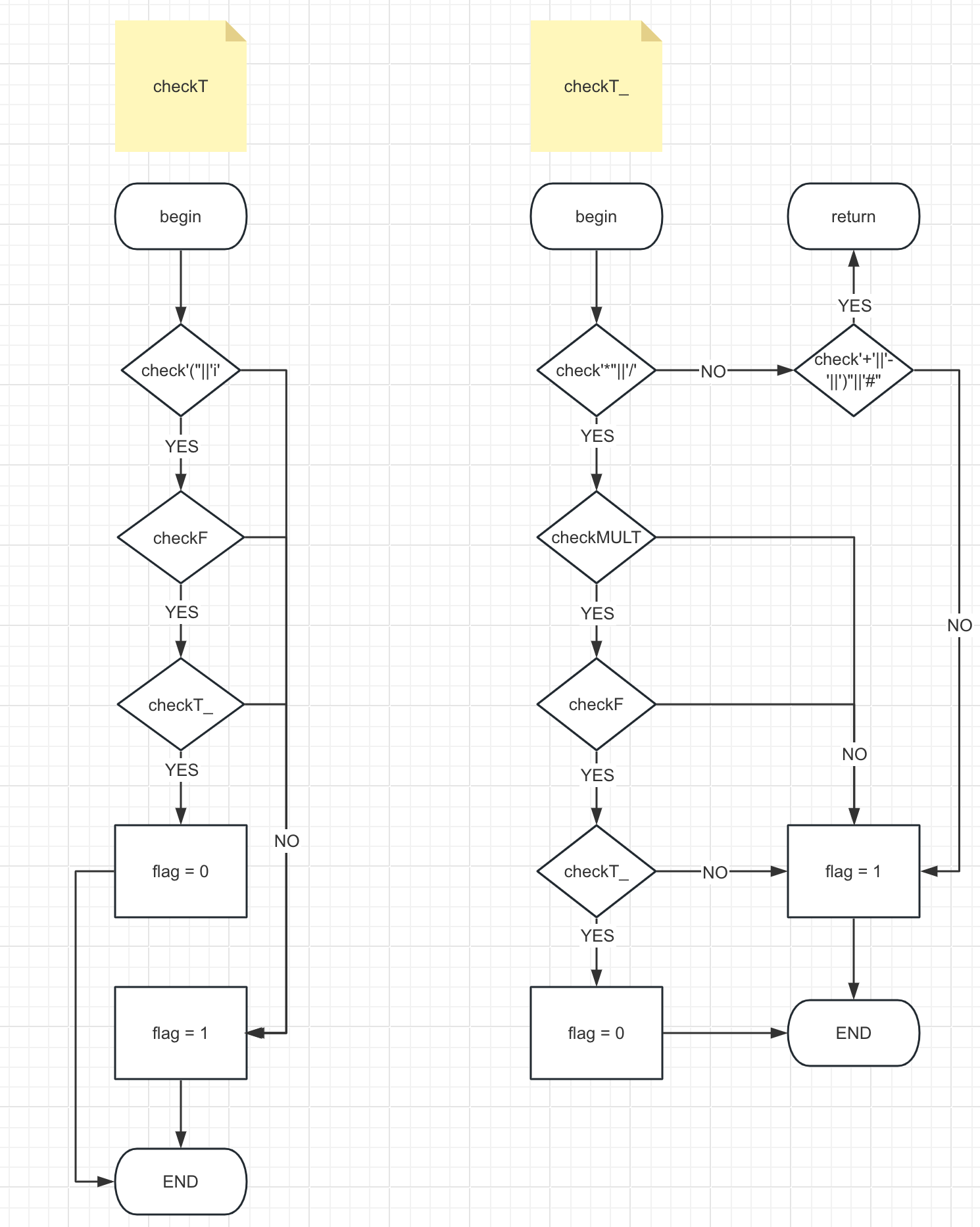
4. 在遇到错误时输出错误信息，将`flag`置为1，停止分析。

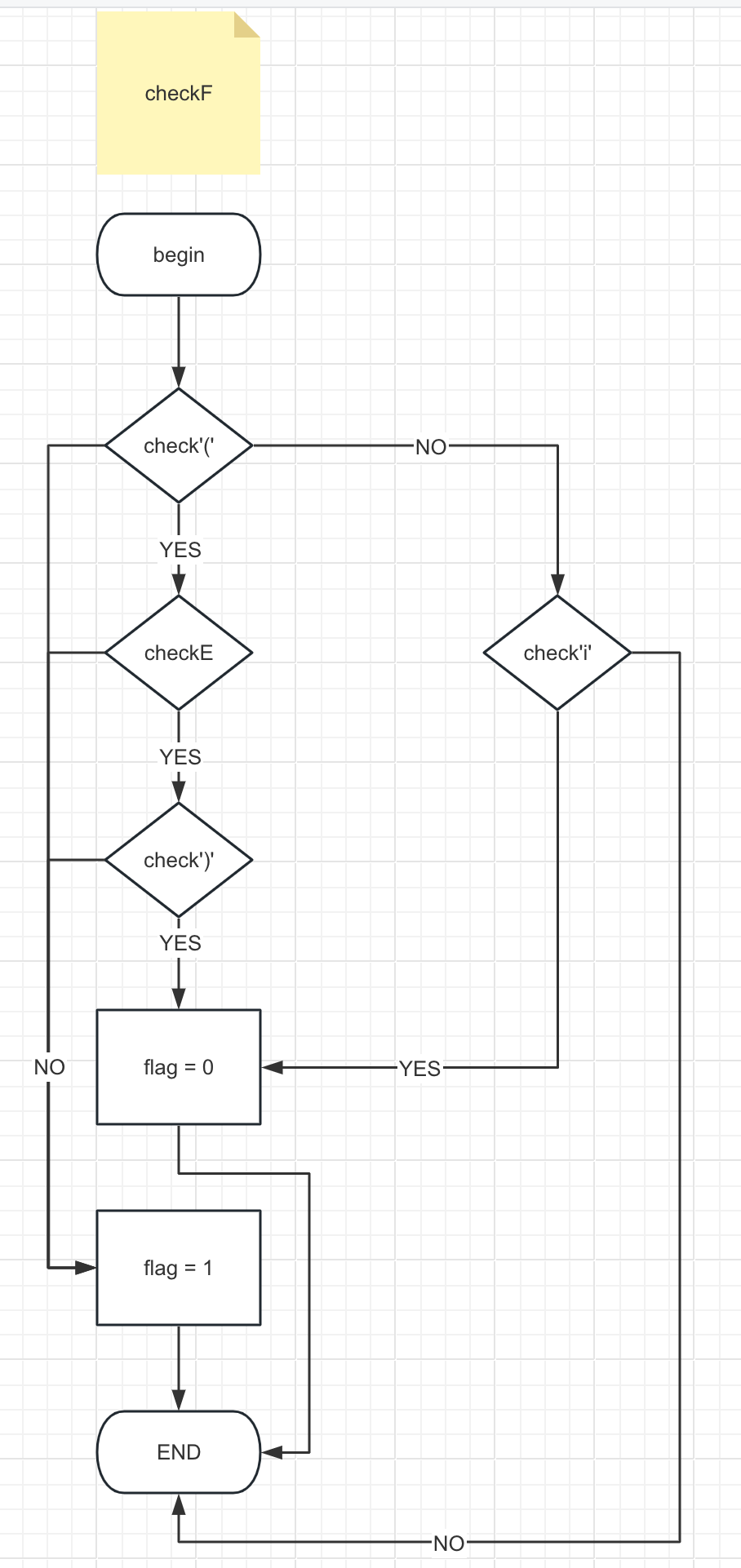
5. 完成分析后，将结果输出到文件"output.txt"，显示"legal"或"illegal"，指示输入字符串是否符合文法规则。

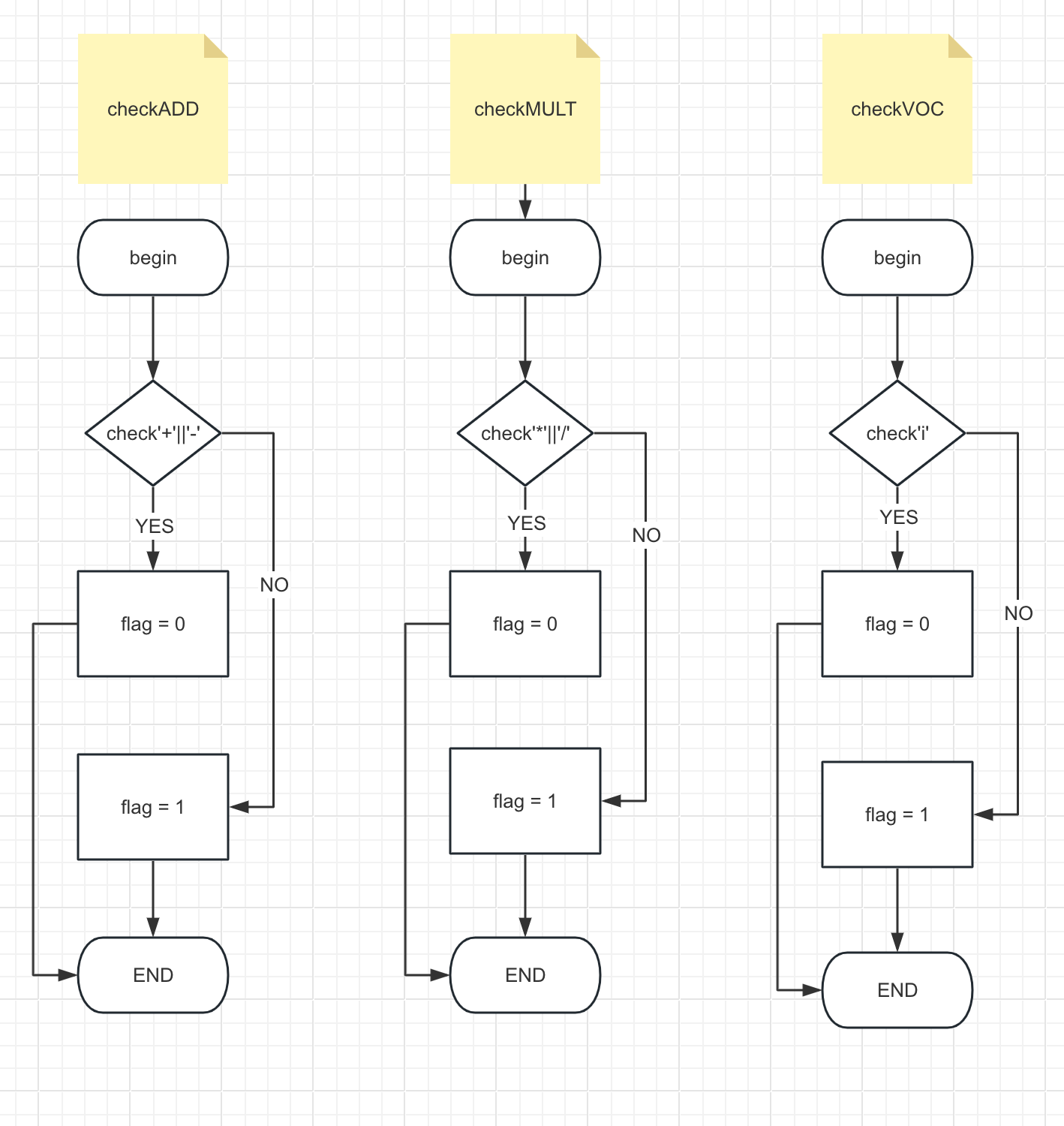
通过递归下降分析方法，程序逐步地检查输入字符串，确保其符合特定的文法规则，同时通过错误处理机制有效地处理语法错误。

程序流程图如图所示：









程序流程图

1. **函数定义及函数间的调用关系**

可参考‘主要数据结构’一栏其中已经详细的描述了函数的内容

1. **程序测试**

测试用例1：

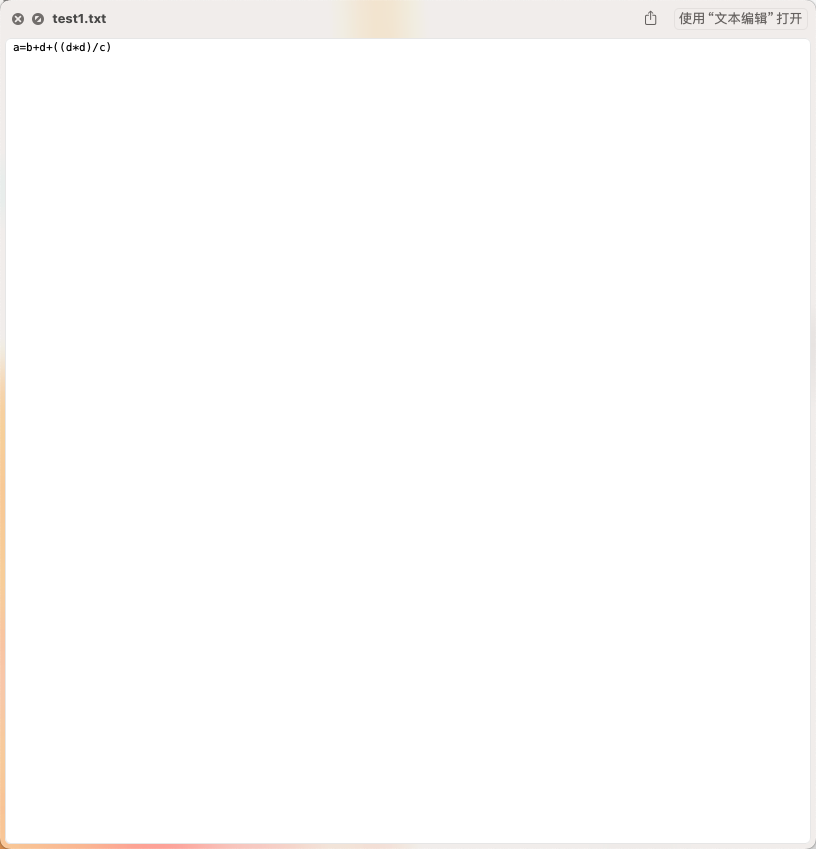
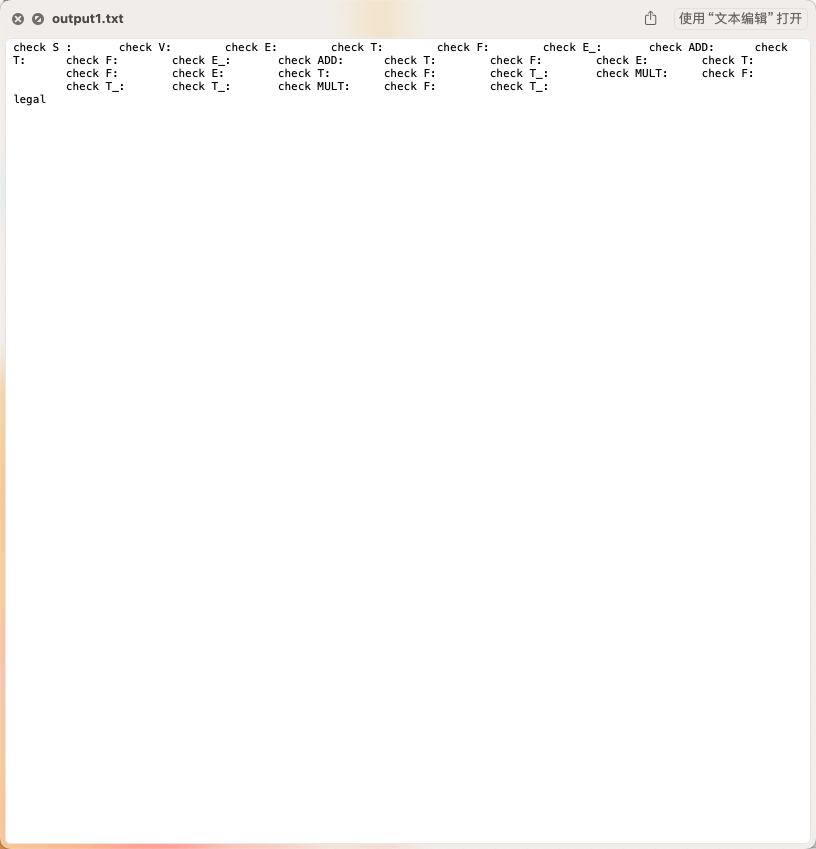


图4.1 测试用例1

程序执行结果：



测试用例2：

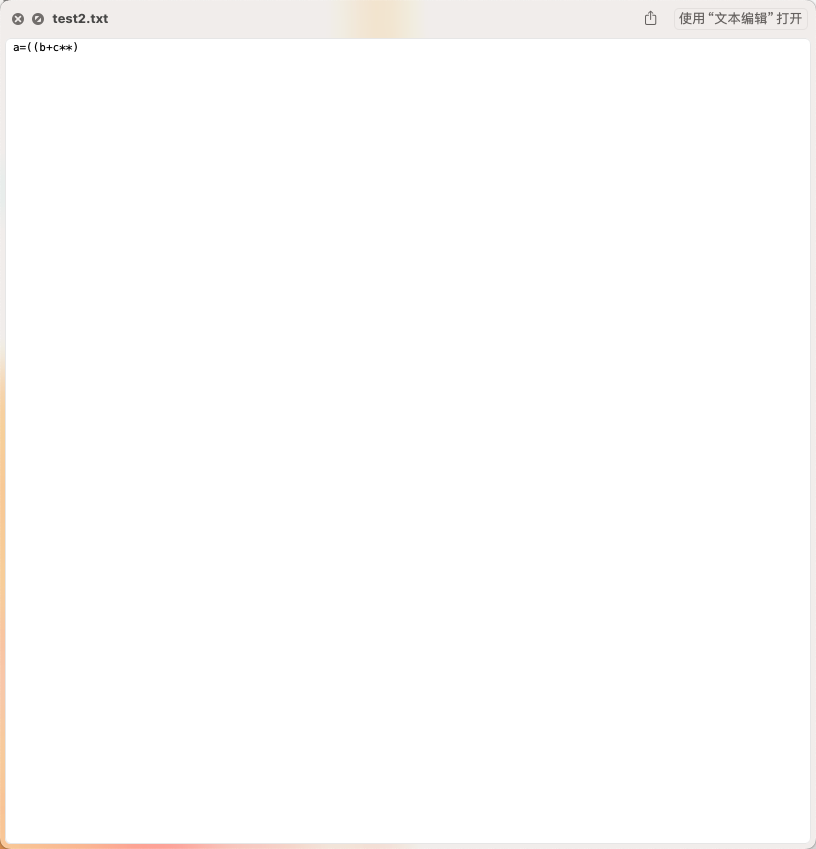


图4.2 测试用例2

程序执行结果：

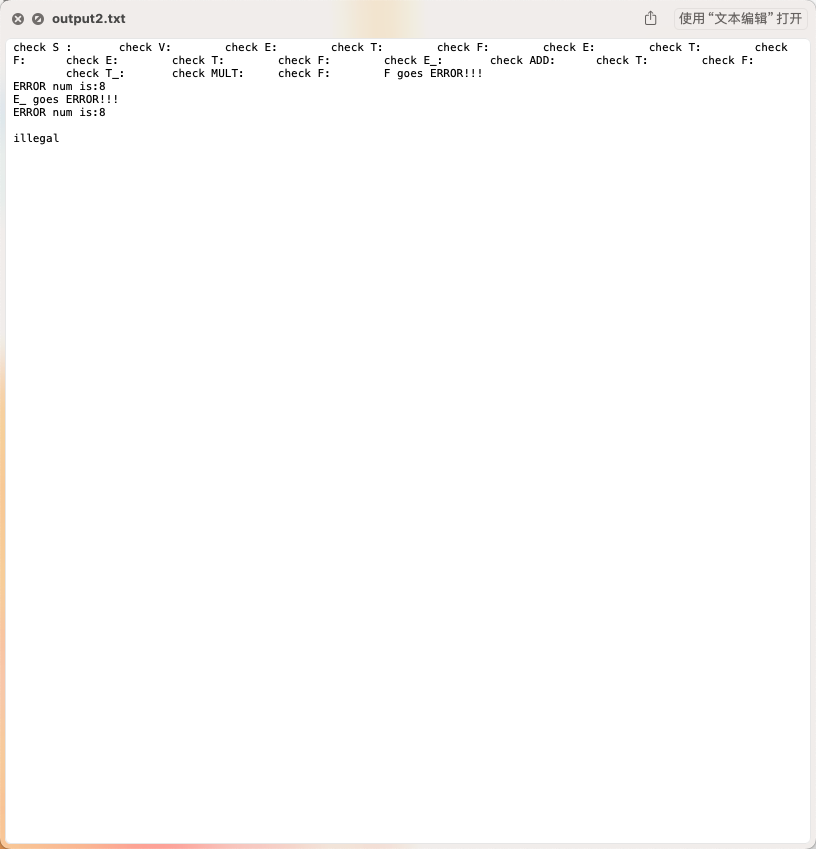


图4.3 测试结果

五．实验结果分析

针对样例1来进行分析：

针对输入字符串"a=b+d+((d\*d)/c)"进行了递归下降分析。以下是程序的执行步骤和分析：

1. `checkS()`: 检查非终结符S。在输入字符串的开始，遇到变量（V）"a"，然后继续检查E。

2. `checkE()`: 检查非终结符E。遇到变量（V）"b"，然后遇到加法操作符（ADD），接着继续检查T。

3. `checkT()`: 检查非终结符T。遇到变量（V）"d"，然后遇到加法操作符（ADD），接着继续检查F。

4. `checkF()`: 检查非终结符F。遇到变量（V）"d"，然后遇到乘法操作符（MULT），继续检查T，然后遇到除法操作符（MULT），再继续检查F。

5. `checkF()`: 继续检查非终结符F。遇到变量（V）"c"，结束。

6. `checkE\_()`: 回溯至非终结符E\_，遇到加法操作符（ADD），继续检查T。

7. `checkT()`: 遇到变量（V）"d"，然后遇到加法操作符（ADD），继续检查F。

8. `checkF()`: 遇到变量（V）"d"，然后遇到乘法操作符（MULT），继续检查T。

9. `checkT()`: 遇到变量（V）"d"，结束。

10. `checkT\_()`: 回溯至非终结符T\_，遇到乘法操作符（MULT），继续检查F。

11. `checkF()`: 遇到变量（V）"c"，结束。

12. `checkT\_()`: 回溯至非终结符T\_，结束。

13. `checkE\_()`: 回溯至非终结符E\_，结束。

14. `checkS()`: 回溯至非终结符S，遇到结束符"#”，语法分析完成。

根据程序的输出，输入字符串符合语法规则，程序输出"legal"表示输入合法。递归下降分析成功地检测到输入字符串的语法结构。

Reference：源代码列表：  
#include<iostream>

#include<cstring>

#include<fstream>

#include<stdlib.h>

using namespace std;

ifstream input("test.txt");

ofstream output("output.txt",ios::app);

string str,s;

int flag = 0,amt = 0;

void checkF();

void checkS();

void checkMUlT();

void checkT\_();

void checkT();

void checkVOC();

void checkADD();

void checkE();

void checkE\_();

void checkT(){

if(flag==0){

output << "check T:" << "\t";

if(s[amt]=='('||s[amt]=='i'){

checkF();

if(flag==0){

if(s[amt]=='\*'||s[amt]=='/'){

checkT\_();

}

else if(s[amt]=='+'||s[amt]=='-'||s[amt]==')'||s[amt]=='#') {

return;

}

else{

flag=1;

output << "T goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

else{

flag=1;

output << "T goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void checkT\_(){

if(flag == 0){

output << "check T\_:"<<'\t';

if(s[amt] == '\*' || s[amt] == '/'){

checkMUlT();

if(flag == 0){

checkF();

if(flag == 0){

checkT\_();

}

}

}

else if(s[amt] == '+' || s[amt] == '-' || s[amt] == ')' || s[amt] == '#'){

return ;

}

else{

flag = 1;

output << "T\_ goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void checkF(){

if(flag == 0){

output << "check F:" << "\t";

if(s[amt] == '('){

amt++ ;

if(s[amt] == '(' || s[amt] == 'i'){

checkE();

if(flag == 0){

if(s[amt] == ')')

amt++;

else{

flag = 1;

output << "F goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

else{

flag = 1;

output << "F goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

else if(s[amt] == 'i')

amt++;

else{

flag = 1;

output << "F goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void checkADD(){

if(flag == 0){

output<<"check ADD:" << "\t";

if(s[amt] == '+' || s[amt] == '-'){

amt++;

}

else{

flag = 1;

output << "ADD goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void checkMUlT(){

if(flag == 0){

output<<"check MULT:" << "\t";

if(s[amt] == '\*' || s[amt] == '/'){

amt++;

}

else{

flag = 1;

output << "MULT goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void checkVOC(){

if(flag == 0){

output<<"check V:" << "\t";

if(s[amt] == 'i'){

amt++;

}

else{

flag = 1;

output << "V goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void checkE(){

if(flag == 0){

output << "check E:" << "\t";

if(s[amt] == '(' || s[amt] == 'i'){

checkT();

if(flag==0){

if(s[amt]=='+'||s[amt]=='-'){

checkE\_();

}

else if(s[amt]==')'||s[amt]=='#'){

return; //rollback

}

else{

flag=1;

output << "E goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

else{

flag=1;

output << "E goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void checkE\_(){

if(flag == 0){

output << "check E\_:" << "\t";

if(s[amt] == '+' || s[amt] == '-'){

checkADD();

if(flag == 0){

if(s[amt] == '(' || s[amt] == 'i'){

checkT();

if(s[amt] == '+' || s[amt] == '-'){

checkE\_();

}

else if(s[amt] == ')' || s[amt] == '#'){

return ;

}

else{

flag = 1;

output << "E\_ goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

}

}

}

void checkS(){

if(flag == 0){

output << "check S :" << "\t";

if(s[amt] == 'i'){

checkVOC();

if(flag == 0 && s[amt] == '='){

amt++;

checkE();

}

else{

flag = 1;

output << "S goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

else{

flag = 1;

output << "S goes ERROR!!!" << "\n";

output << "ERROR num is:" << amt << endl;

}

}

}

void fcheck(){

if(s[0] == '#'){

output<< "system stop!!!" << "\n";

}

}

int main(){

if(input.is\_open()){

getline(input,str);

input.close();

s = str;

//output << s.length() << endl;

}

int len = s.length();

s += '#';

for(int i = 0;i < len;i++){

if(isalpha(s[i]))

s[i] = 'i';

}

//output<< str << "\n";

fcheck();

checkS();

output << "\n";

if(flag == 0){

output << "legal" << "\n";

}

if(flag == 1){

output << "illegal" << "\n";

}

cout << "program complete!" << "\n";

output.close();

return 0;

}