**北 京 邮 电 大 学**

**实 验 报 告**

**课程名称\_\_\_\_**编译原理与技术**\_\_\_\_\_\_**

**实验名称\_\_**词法分析器程序的设计与实现**\_\_**

**\_**计算机**\_学院\_**10**\_班 姓名\_**许子康**\_**

**教师\_\_\_\_\_\_\_\_ 成绩\_\_\_\_\_\_**

**\_**2020**\_年\_**10**\_月\_**25**\_日**

一、实验目的

利用编译原理课程所学知识，利用C++语言编写出一个完整的词法分析器，进一步掌握词法分析器的内容和编译原理课程所学内容。

二、实验环境

编程语言：C++

编程工具：Clion

编程环境：Windows10

三、实验要求

用C++语言设计一个C语言词法分析程序，要求改词法分析程序满足以下要求：

（1）可以识别出用C语言编写的源程序中的每一个单词符号，并以记号的形式输出每个单词符号。

（2）可以识别并读取源程序中的注释。

（3）可以统计源程序中的渔具行数、单词个数和字符个数，其中空格和回车不计算为单词，并输出统计结果。

（4）检查源程序中存在的非法字符错误，并可以报告错误所在的行位置。

（5）对源程序中出现的错误进行适当的回复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有错误。

四、实验分析

1、语言分析

C语言使用的词汇可以分为标识符、关键字、无符号数、关系运算符、算数运算符、逻辑运算符、标点符号、注释标记、分隔符和其他符号。

1）标识符：由数字和字母以及下划线”\_”组成，由字母和“\_”开头的用户自定义的符号串。

2）关键字：C语言32个关键字：auto、double、int、struct、break、else、long、switch、case、enum、register、typedef、char、extern、return、union、const、float、short、unsigned、continue、for、signed、void、defalut、goto、sizeof、volatil、do、if、while、static。

3）无符号数：由整数部分、可选的小数部分和可选的指数部分组成，如3、3E4、3.14、2.3E-3、2.4E+6等。

4）关系运算符：>、<、==、!=、<=、>=。

5）算术运算符：+、++、+=、-、--、-=、\*、\*=、/、/=、%、%=、=、<<、>>等。

6）逻辑运算符：&&、&、||、|、！、^、~等。

7）标点符号：[、]、{、}、（、）、“、”、‘、’等。

8）注释标记：以“/\*”开始，以“\*/”结束或者”//”之后的一行。

9）分隔符：单词符号间的分割符如空格、‘/t’等。

10）其他符号：->、#、等。

2、正规文法

1）标识符的文法

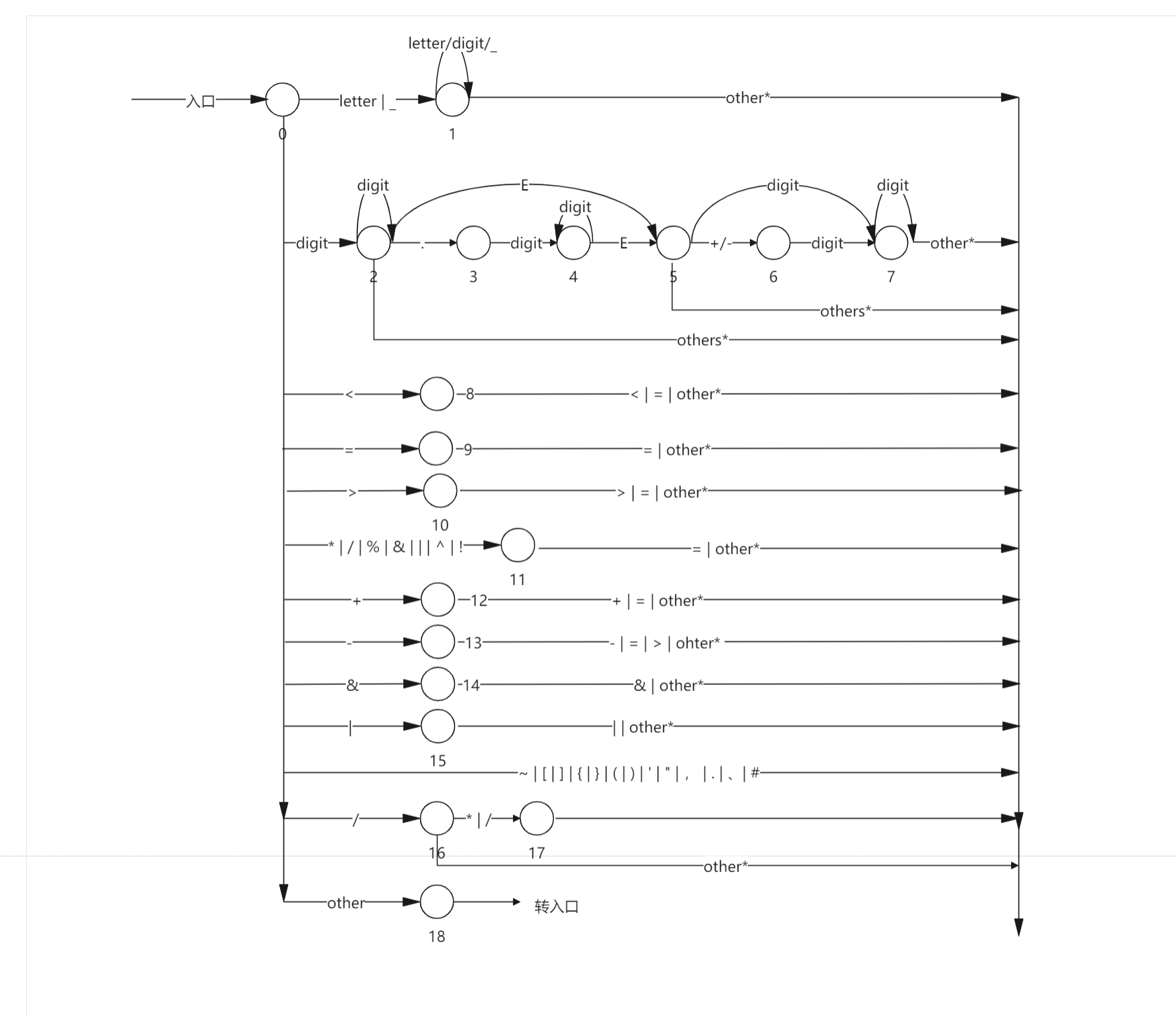
2）无符号数的文法

3) 关系运算符文法

4）算术、逻辑运算符和标点符号文法及其它符号文法

5）注释头符号的文法

3、状态转移图



4、词法分析器的构造

有了上述状态转换图，只要把语义动作进一步添加到状态转换图中，使每一个状态都对应一小段程序，就可以构造出相应的词法分析程序。如：

在开始状态，首先要读进一个字符。若读入的字符是一个空格（包括blank、tab、enter）就跳过它，继续读字符，直到读进一个非空字符为止，接下来的工作就是根据所读进的非空字符转相应的程序段进行处理。

在标识符状态，识别并组合出一个标识符之后，还必须加入一些动作，如查关键字表，以确定识别出的单词符号是关键字还是用户自定义标识符，并输出相应的记号。

在无符号数状态，可识别出各种常数，包括整数、小数和无符号数。在组合常数的同时，还要进行从字符串到数字的转换。

在“<”状态，若读进的下一个字符是“=”，则输出关系运算符“<=”；若读进的下一个字符是“<”，则输出关系运算符“<<”；否则输出关系运算符“<”。

在“>”状态，若读进的下一个字符是“=”，则输出关系运算符“>=”；若读进的下一个字符是“>”，则输出关系运算符“>>”；否则输出关系运算符“<”。

在“+”状态，若读进的下一个字符是“+”，则输出运算符“++”；若读进的下一个字符是“=”，则输出运算符“+=”；否则输出运算符“+”。

在“-”状态，若读进的下一个字符是“-”，则输出运算符“--”；若读进的下一个字符是“=”，则输出运算符“-=”；否则输出运算符“-”。

在“&”状态，若读进的下一个字符是“&”，则输出运算符“&&”；若读进的下一个字符是“=”，则输出运算符“&=”；否则输出运算符“&”。

在“|”状态，若读进的下一个字符是“|”，则输出运算符“||”；若读进的下一个字符是“=”，则输出运算符“|=”；否则输出运算符“|”。

在“\* / / / / % / ! / ^ / =”状态，若读进的下一个字符是“=”，则输出符号“\*= / /= / %= / != / ^= / ==”；否则，输出“\* / / / / % / ! / & / | / ! / ^ / =”。

在“/”状态，若读进的下一个字符是“\*”（或者“/”），则进入注释处理状态，词法分析程序要做的工作是跳过注释，具体做法就是不断地读字符，直到遇见“\*/”（或者回车），然后转开始状态，继续识别和分析下一个单词；若读进地下一个字符不是“\*”，则输出“/”。

在其他算术运算符和标点符号状态，只需输出其相应的记号即可。

若进入错误处理状态，表示词法分析程序从源程序中读入了一个不合法地字符。所谓不合法的字符是指该语言不包括以此字符开头的单词符号。词法分析程序发现不合法字符时，要做错误处理，其主要工作是显示或打印错误信息，并跳过这个字符，然后转开始状态继续识别和分析下一个单词符号。

还有一点应该注意，在词法分析过程中，为了判断是否已经读到单词符号的右端字符，有时需要向前多读入一个字符，比如在标识符状态和无符号数状态，因此词法分析程序在返回调用程序之前，应将向前指针后退一个字符。

5、输出形式

对识别出的记号以二元式的形式加以输出，形式为<记号，属性>。

|  |  |  |
| --- | --- | --- |
| 正规表达式 | 记号 | 属性 |
| auto | auto | - |
| double | double | - |
| int | int | - |
| struct | struct | - |
| break | break | - |
| else | else | - |
| long | long | - |
| switch | switch | - |
| case | case | - |
| enum | enum | - |
| register | register | - |
| typdef | typdef | - |
| char | char | - |
| extern | extern | - |
| return | return | - |
| union | union | - |
| const | const | - |
| float | float | - |
| short | short | - |
| unsigned | unsigned | - |
| continue | continue | - |
| for | for | - |
| signed | signed | - |
| void | void | - |
| defalut | defalut | - |
| goto | goto | - |
| sizeof | sizeof | - |
| volatile | volatile | - |
| do | do | - |
| if | if | - |
| while | while | - |
| static | static | - |
| id | id | 符号表入口 |
| num | num | 常数值 |
| < | relop | LT |
| <= | relop | LE |
| == | relop | EQ |
| > | relop | GT |
| >= | relop | GE |
| != | relop | NE |
| = | = | - |
| + | + | - |
| ++ | ++ | - |
| += | += | - |
| - | - | - |
| -- | -- | - |
| -= | -= | - |
| \* | \* | - |
| \*= | \*= | - |
| / | / | - |
| /= | /= | - |
| % | % | - |
| %= | %= | - |
| & | & | - |
| && | && | - |
| &= | &= | - |
| | | | | - |
| || | || | - |
| |= | |= | - |
| << | << | - |
| >> | >> | - |
| ^ | ^ | - |
| ^= | ^= | - |
| [ | [ | - |
| ] | ] | - |
| { | { | - |
| } | } | - |
| ( | ( | - |
| ) | ) | - |
| ； | ； | - |
| ， | ， | - |
| -> | -> | - |
| # | # | - |
| ‘ | ‘ | - |
| “ | “ | - |
| : | : | - |

五、变量解释

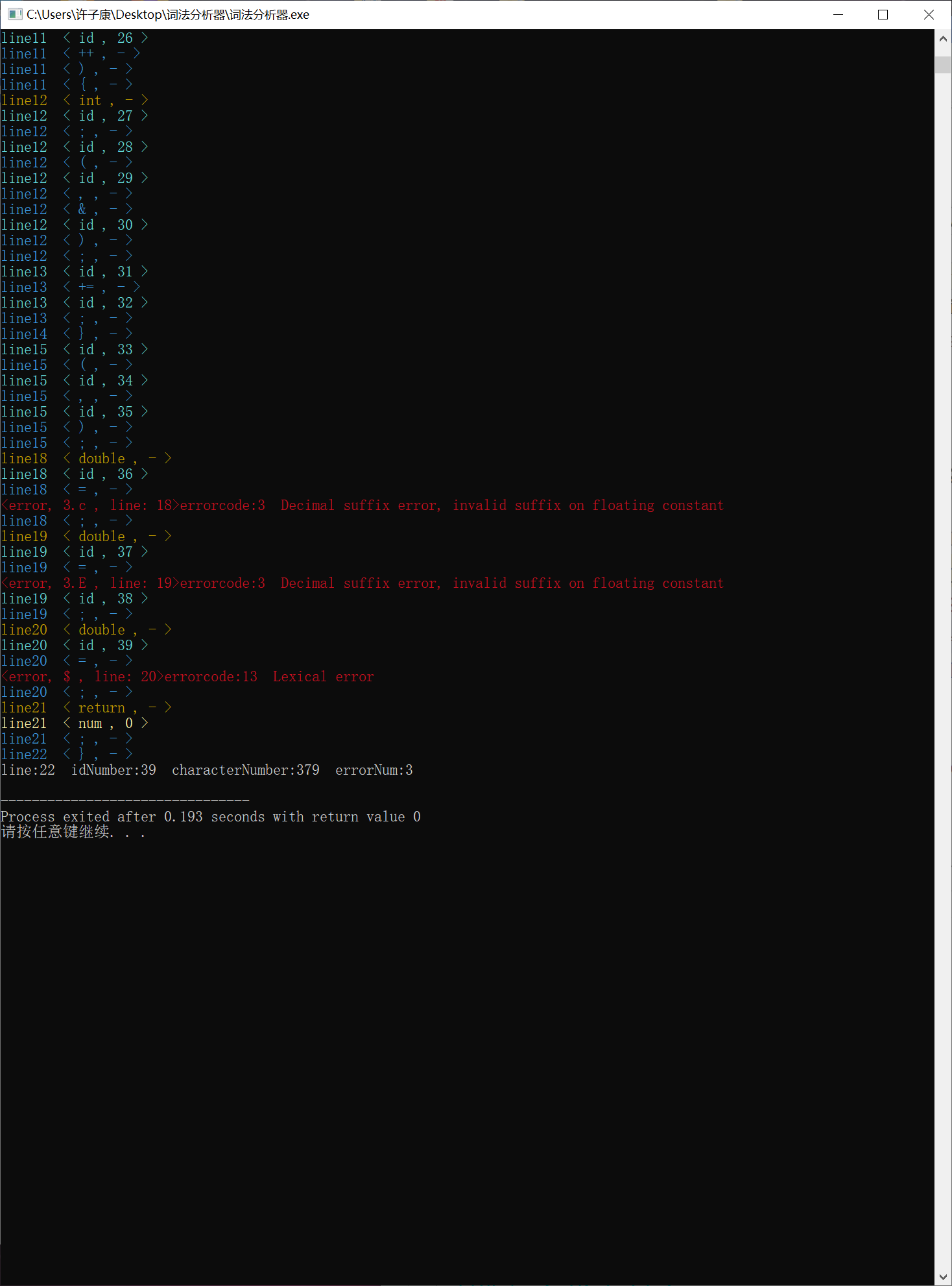
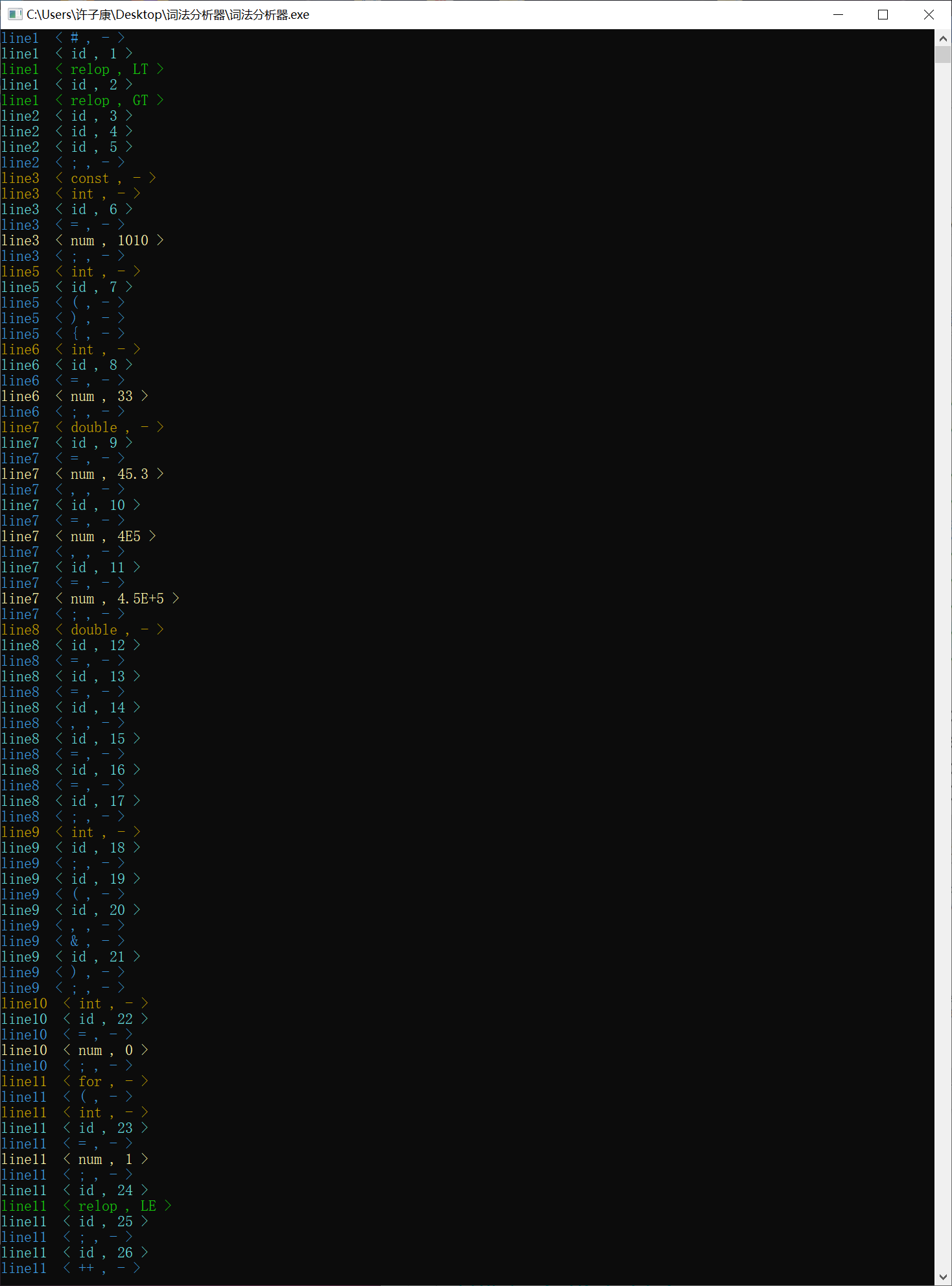
*//单词符号对应类型：1标识符，2关键字，3无符号数，4关系运算符，5算术运算符和标点符号*#define **idType 1**#define **keywordsType 2**#define **numType 3**#define **relopType 4**#define **singleType 5**const int maxn = **1010**;  
int state; *//整型变量，当前状态指示*int len;*//正在识别的单词字符串长度*char token[**100**]; *//字符数组，存放当前正在识别的单词字符串*char C; *//字符变量，存放当前读入的字符*int iskey; *//整型变量，值为-1，标识识别出的单词是用户自定义标识符，否则，表示识别出的单词是关键字，其值为关键字的记号*int identry; *//整型变量，单词在符号表中的指针位置*FILE \*sptr;*//源码文件*char keywords[**100**][**100**];*//关键字字符数组*int numofKeywords;*//关键字数量*int idNum;*//标识符数量*char idNumStr[**100**];*//用于存储标识符id值的字符串*struct MarkTable{*//记号：标志和属性* int **type**;*//类型，1标识符，2关键字，3无符号数，4关系运算符，5算术运算符，6标点符号，7赋值号，8注释标记，9分隔符* char **mark**[**100**];  
 char **val**[**100**];  
 char **str**[**100**];*//存储原token*}mark[**40000**];  
int numofMark;*//记号数量*int lineNum = **1**;*//文章行数*int charNum;*//文章字母数*bool judgeError;*//判断是否出现错误*int errorType;*//错误类型*int errorNum;*//错误数量*

六、实验程序及结果

测试程序如下：

#include <cstdio>  
using namespace std;  
const int maxn = **1010**;  
  
int main(){  
 int a = **33**;  
 double b = **45.3**, c = **4E5**, d = **4.5E+5**;  
 double add1 = a + b, add2 = c + d;  
 int n; scanf("%d", &n);  
 int tot = **0**;  
 for(int i = **1**; i <= n; i++){  
 int val; scanf("%d", &val);  
 tot += val;  
 }  
 printf("%d**\n**", tot);  
 */\*error\*/  
 //error* double error1 = **3.**c;  
 double error2 = 3.Ed;  
 double error3 = $;  
 return **0**;  
}

词法分析器分析上述测试程序得到结果如下：



七、源代码

源码如下：

#include <windows.h>  
#include <bits/stdc++.h>  
using namespace std;  
#define **LL** long long  
#define **Inf 1e9***//单词符号对应类型：1标识符，2关键字，3无符号数，4关系运算符，5算术运算符和标点符号*#define **idType 1**#define **keywordsType 2**#define **numType 3**#define **relopType 4**#define **singleType 5**const int maxn = **1010**;  
int state; *//整型变量，当前状态指示*int len;*//正在识别的单词字符串长度*char token[**100**]; *//字符数组，存放当前正在识别的单词字符串*char C; *//字符变量，存放当前读入的字符*int iskey; *//整型变量，值为-1，标识识别出的单词是用户自定义标识符，否则，表示识别出的单词是关键字，其值为关键字的记号*int identry; *//整型变量，单词在符号表中的指针位置*FILE \*sptr;*//源码文件*FILE \*dptr;*//结果文件*char keywords[**100**][**100**];*//关键字字符数组*int numofKeywords;*//关键字数量*int idNum;*//标识符数量*char idNumStr[**100**];*//用于存储标识符id值的字符串*struct MarkTable{*//记号：标志和属性* int **type**;*//类型，1标识符，2关键字，3无符号数，4关系运算符，5算术运算符，6标点符号，7赋值号，8注释标记，9分隔符* char **mark**[**100**];  
 char **val**[**100**];  
 char **str**[**100**];*//存储原token*}mark[**40000**];  
int numofMark;*//记号数量*int lineNum = **1**;*//文章行数*int charNum;*//文章字母数*bool judgeError;*//判断是否出现错误*int errorType;*//错误类型*int errorNum;*//错误数量  
/\*  
 \* 过程，每调用一次，根据向前指针forward的指示从输入缓冲区中读一个字符，并把它放入变量C中，然后，移动forwar，使之指向下一个字符  
 \*/*void get\_char(){  
 C = fgetc(sptr);  
 if(charNum != **EOF**) ++charNum;  
}  
*/\*  
 \* 过程，每次调用时，检查C中字符是否为空格，如果是，则反复调用过程get\_char，直到C中进入一个非空字符为止  
 \*/*bool get\_nbc(){  
 bool tag = false;  
 while(C == ' ' || C == '**\n**' || C == '**\t**'){  
 if(C == '**\n**'){  
 lineNum++;  
 tag = true;  
 }  
 C = fgetc(sptr);  
 }  
 return tag;  
}  
*/\*  
 \* 布尔函数，判断C中的字符是否是字母，若是则返回true，否则返回false  
 \*/*bool letter(){  
 if((C >= 'A' && C <= 'Z') || (C >= 'a' && C <= 'z')) return true;  
 return false;  
}  
*/\*  
 \* 布尔函数，判断C中的字符是否为数字，若是则返回true，否则返回false  
 \*/*bool digit(){  
 if(C >= '0' && C <= '9') return true;  
 return false;  
}  
*/\*  
 \*过程，把C中的字符连接在token中的字符串后面  
 \*/*void cat(){  
 token[strlen(token)] = C;  
}  
*/\*  
 \*过程，向前指针forward后退一个字符  
 \*/*void retract(){  
 fseek(sptr, -**1**, **SEEK\_CUR**);  
}  
*/\*  
 \*函数，根据token中单词查关键字表，若token中单词是关键字，则返回该关键字的记号，否则，返回值为-1  
 \*/*int reserve(){  
 int tag = -**1**;  
 for(int i = **1**; i <= numofKeywords; i++){  
 if(strcmp(keywords[i], token) == **0**){  
 tag = i;  
 break;  
 }  
 }  
 return tag;  
}  
void color(short x) *//自定义函根据参数改变颜色*{  
 if(x>=**0** && x<=**15**)*//参数在0-15的范围颜色* SetConsoleTextAttribute(GetStdHandle(**STD\_OUTPUT\_HANDLE**), x); *//只有一个参数，改变字体颜色* else*//默认的颜色白色* SetConsoleTextAttribute(GetStdHandle(**STD\_OUTPUT\_HANDLE**), **7**);  
}  
*/\*  
 \*过程，对发现的错误进行相应的处理  
 \*/*void outputError(){  
 color(**4**);  
 judgeError = false;  
 errorNum++;  
 printf("<error, %s , line: %d>", token, lineNum);  
 fprintf(dptr, "<error, %s , line: %d>", token, lineNum);  
  
  
 if(errorType == **3** || errorType == **5** || errorType == **6**){  
 printf("errorcode:%d Decimal suffix error, invalid suffix on floating constant**\n**", errorType);  
 fprintf(dptr, "errorcode:%d Decimal suffix error, invalid suffix on floating constant**\n**", errorType);  
 }  
 else if(errorType == **13**){  
 printf("errorcode:%d Lexical error**\n**", errorType);  
 fprintf(dptr, "errorcode:%d Lexical error**\n**", errorType);  
 }  
 else if(errorType == **23**){  
 printf("errorcode:%d missing tereminating **\"** character**\n**", errorType);  
 fprintf(dptr, "errorcode:%d missing tereminating **\"** character**\n**", errorType);  
 }  
 else if(errorType == **24**){  
 printf("errorcode:%d missing tereminating ' character**\n**", errorType);  
 fprintf(dptr, "errorcode:%d missing tereminating ' character**\n**", errorType);  
 }  
}  
void error(int type){  
 if(type != **23** && type != **24**) cat();  
 errorType = type;  
 judgeError = true;  
 outputError();  
 *//cout << "There is a bug!" << endl;*}  
void LoadKeywords(){*//关键字的导入* ifstream ifs;  
 ifs.open("Keywords.txt", ios::**in**);  
 if(!ifs.is\_open()){  
 color(**4**);  
 cout << "关键字文件不存在!" << endl;  
 ifs.close();  
 return;  
 }  
 ifs >> numofKeywords;  
 for(int i = **1**; i <= numofKeywords; i++)  
 ifs >> keywords[i];  
 ifs.close();  
}  
void retMark(MarkTable nowMark){*//输出记号  
  
 /\*if(nowMark.type == idType) printf("idType");  
 else if(nowMark.type == keywordsType) printf("keywordsType");  
 else if(nowMark.type == numType) printf("NumType");  
 else if(nowMark.type == relopType) printf("relopType");  
 else if(nowMark.type == singleType) printf("singleType");\*/* if(nowMark.**type** == **idType**) color(**11**);  
 else if(nowMark.**type** == **keywordsType**) color(**6**);  
 else if(nowMark.**type** == **numType**) color(**14**);  
 else if(nowMark.**type** == **relopType**) color(**10**);  
 else if(nowMark.**type** == **singleType**) color(**3**);  
 printf("line%d ",lineNum);  
 printf("< %s , %s >**\n**", nowMark.**mark**, nowMark.**val**);  
 fprintf(dptr, "line%d ",lineNum);  
 fprintf(dptr, "< %s , %s >**\n**", nowMark.**mark**, nowMark.**val**);  
  
}  
void Work();  
int main(){  
 LoadKeywords();  
 Work();  
 return **0**;  
}  
void Work(){  
 bool isNextLine;  
 sptr = fopen("code.txt", "r");  
 dptr = fopen("markTable.txt", "w+");  
 state = **0**;  
 do{  
 switch(state){  
 case **0**:*//初始状态* if(judgeError == false) memset(token, **0**, sizeof(token));  
 get\_char();  
 isNextLine = get\_nbc();  
 *//printf("%d (%c) %s\n",isNextLine, C,token);  
 /\*if(isNextLine){  
 state = 24;  
 break;  
 }\*/* switch(C){  
 case 'a':  
 case 'b':  
 case 'c':  
 case 'd':  
 case 'e':  
 case 'f':  
 case 'g':  
 case 'h':  
 case 'i':  
 case 'j':  
 case 'k':  
 case 'l':  
 case 'm':  
 case 'n':  
 case 'o':  
 case 'p':  
 case 'q':  
 case 'r':  
 case 's':  
 case 't':  
 case 'u':  
 case 'v':  
 case 'w':  
 case 'x':  
 case 'y':  
 case 'z':  
 case 'A':  
 case 'B':  
 case 'C':  
 case 'D':  
 case 'E':  
 case 'F':  
 case 'G':  
 case 'H':  
 case 'I':  
 case 'J':  
 case 'K':  
 case 'L':  
 case 'M':  
 case 'N':  
 case 'O':  
 case 'P':  
 case 'Q':  
 case 'R':  
 case 'S':  
 case 'T':  
 case 'U':  
 case 'V':  
 case 'W':  
 case 'X':  
 case 'Y':  
 case 'Z': state = **1**; break; *//设置标识符状态* case '0':  
 case '1':  
 case '2':  
 case '3':  
 case '4':  
 case '5':  
 case '6':  
 case '7':  
 case '8':  
 case '9': state = **2**; break; *//设置常数符状态* case '<': state = **8**; break; *//设置'<'符状态* case '>': state = **9**; break; *//设置'>'符状态* case ':': state = **10**; break;*//设置':'符状态* case '/': state = **11**; break;*//设置'/'符状态* case '=': state = **15**; break;*//设置'='符状态* case '+': state = **16**; break;*//设置'+'符状态* case '-': state = **17**; break;*//设置'-'符状态* case '\*': state = **18**; break;*//设置'\*'符状态* case '%': state = **19**; break;*//设置'%'符状态* case '!': state = **20**; break;*//设置'!'符状态* case '&': state = **21**; break;*//设置'&'符状态* case '|': state = **22**; break;*//设置'|'符状态* case '"': state = **23**; break;*//设置'"'符状态* case '**\'**': state = **24**; break;  
 case '^':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "^");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '.':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, ".");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case ',':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, ",");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '~':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "~");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '?':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "?");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '(':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "(");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case ')':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, ")");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '{':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "{");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '}':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "}");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '[':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "[");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case ']':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "]");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case ';':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, ";");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '#':{  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "#");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 default: state = **13**; break;  
 }  
 break;  
 case **1**:*//标识符状态* cat();  
 get\_char();  
 if(letter() || digit() || C == '\_') state = **1**;  
 else {  
 retract();  
 state = **0**;  
 iskey = reserve();*//关键字返回记号，否则返回-1* if(iskey == -**1**) {*//不是关键字* idNum++;  
 itoa(idNum, idNumStr, **10**);  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
 nowMark.**type** = **idType**;  
  
 strcpy(nowMark.**mark**, "id");  
 strcpy(nowMark.**val**, idNumStr);  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }else {*//关键字* numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
 nowMark.**type** = **keywordsType**;  
 strcpy(nowMark.**mark**, token);  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 }  
 break;  
 case **2**:*//常数状态* cat();  
 get\_char();  
 switch(C){  
 case '0':  
 case '1':  
 case '2':  
 case '3':  
 case '4':  
 case '5':  
 case '6':  
 case '7':  
 case '8':  
 case '9': state = **2**; break;  
 case '.': state = **3**; break;  
 case 'E': state = **5**; break;  
 default:  
 retract();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **numType**;  
 strcpy(nowMark.**mark**, "num");  
 strcpy(nowMark.**val**, token);  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 break;  
 case **3**:*//小数点状态* cat();  
 get\_char();  
 if(digit()) state = **4**;  
 else {  
 error(**3**);  
 state = **0**;  
 }  
 break;  
 case **4**:*//小数状态* cat();  
 get\_char();  
 switch(C){  
 case '0':  
 case '1':  
 case '2':  
 case '3':  
 case '4':  
 case '5':  
 case '6':  
 case '7':  
 case '8':  
 case '9': state = **4**; break;  
 case 'E': state = **5**; break;  
 default:*//识别出小数* retract();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **numType**;  
 strcpy(nowMark.**mark**, "num");  
 strcpy(nowMark.**val**, token);  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 break;  
 case **5**:*//指数状态* cat();  
 get\_char();  
 switch(C){  
 case '0':  
 case '1':  
 case '2':  
 case '3':  
 case '4':  
 case '5':  
 case '6':  
 case '7':  
 case '8':  
 case '9': state = **7**; break;  
 case '+':  
 case '-': state = **6**; break;  
 default:  
 retract();  
 error(**5**);  
 state = **0**;  
 break;  
 }  
 break;  
 case **6**:  
 cat();  
 get\_char();  
 if(digit()) state = **7**;  
 else {  
 retract();  
 error(**6**);  
 state = **0**;  
 }  
 break;  
 case **7**:  
 cat();  
 get\_char();  
 if(digit()) state = **7**;  
 else {  
 retract();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **numType**;  
 strcpy(nowMark.**mark**, "num");  
 strcpy(nowMark.**val**, token);  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);*//返回无符号数* break;  
 }  
 break;  
 case **8**:*//'<'状态* cat();  
 get\_char();  
 switch(C){  
 case '=':{*//<=* cat();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **relopType**;  
 strcpy(nowMark.**mark**, "relop");  
 strcpy(nowMark.**val**, "LE");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '>':{*//<>* cat();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **relopType**;  
 strcpy(nowMark.**mark**, "relop");  
 strcpy(nowMark.**val**, "NE");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case '<':{*//<<* cat();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "<<");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 default:{*//<* retract();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **relopType**;  
 strcpy(nowMark.**mark**, "relop");  
 strcpy(nowMark.**val**, "LT");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
  
 }  
 break;  
 case **9**:*//'>'状态* cat();  
 get\_char();  
 *//printf("<<%c>>", C);* if(C == '='){*//>=* cat();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **relopType**;  
 strcpy(nowMark.**mark**, "relop");  
 strcpy(nowMark.**val**, "GE");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else if(C == '>'){  
 cat();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, ">>");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else {*//>* retract();  
 state = **0**;  
  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **relopType**;  
 strcpy(nowMark.**mark**, "relop");  
 strcpy(nowMark.**val**, "GT");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **10**:  
 cat();  
 get\_char();  
 if(C == '='){*//：=* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, ":=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }else {*//:* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, ":");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **11**:*// /* cat();  
 get\_char();  
 if(C == '\*') state = **12**; *///\** else if(C == '/') state = **14**; *////* else if(C == '='){  
 cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "/=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else {  
 retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "/");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **12**:*//注释处理，/\** get\_char();  
 if(C == '**\n**') lineNum++;  
 while(C != '\*') {  
 get\_char();  
 if(C == '**\n**') lineNum++;  
 }  
 get\_char();  
 if(C == '**\n**') lineNum++;  
 if(C == '/') state = **0**;  
 else state = **12**;  
 break;  
 case **14**:*//注释处理，//* get\_char();  
 while(C != '**\n**' && C != **EOF**) {  
 get\_char();  
 }  
 if(C == '**\n**') lineNum++;  
 state = **0**;  
 break;  
 case **15**:*//=* cat();  
 get\_char();  
 if(C == '='){*//==* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **relopType**;  
 strcpy(nowMark.**mark**, "relop");  
 strcpy(nowMark.**val**, "EQ");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else {*//=* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **16**:*//+状态* cat();  
 get\_char();  
 if(C == '='){*//+=* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "+=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else if(C == '+'){*//++* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "++");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else {*//+* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **17**:*//-状态* cat();  
 get\_char();  
 if(C == '='){*//-=* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "-=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else if(C == '-'){*//--* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "--");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else if(C == '>'){*//->* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "->");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else {*//-* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "-");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **18**:*//\*状态* cat();  
 get\_char();  
 if(C == '='){*//\*=* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "\*=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
  
 else {*//\** retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "\*");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **19**:*//%状态* cat();  
 get\_char();  
 if(C == '='){*//%=* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "%=");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
  
 else {*//%* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "%");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **20**:*//!* cat();  
 get\_char();  
 if(C == '='){*//!=* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **relopType**;  
 strcpy(nowMark.**mark**, "relop");  
 strcpy(nowMark.**val**, "NE");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
  
 else {*//!* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "!");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **21**:*//&* cat();  
 get\_char();  
 if(C == '&'){*//&&* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "&&");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else {*//&* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "&");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **22**:*//|* cat();  
 get\_char();  
 if(C == '|'){*//||* cat();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "||");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 else {*//&* retract();  
 state = **0**;  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
  
 nowMark.**type** = **singleType**;  
 strcpy(nowMark.**mark**, "|");  
 strcpy(nowMark.**val**, "-");  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 }  
 break;  
 case **23**: {*//"* cat();  
 get\_char();  
 bool tag = false;  
 if(C == '**\n**') {  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 if(C == '**\\**') {  
 cat();  
 get\_char();  
 if(C == '**\n**'){  
 tag = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 cat();  
 get\_char();  
 if(C == '**\n**'){  
 tag = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 }  
 while(C != '"') {  
 cat();  
 get\_char();  
 if(C == '**\n**') {  
 tag = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 if(C == '**\\**') {  
 cat();  
 get\_char();  
 if(C == '**\n**'){  
 tag = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 cat();  
 get\_char();  
 if(C == '**\n**'){  
 tag = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 }  
 }  
 if(tag) break;  
 cat();  
 state = **0**;  
  
 idNum++;  
 itoa(idNum, idNumStr, **10**);  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
 nowMark.**type** = **idType**;  
  
 strcpy(nowMark.**mark**, "id");  
 strcpy(nowMark.**val**, idNumStr);  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case **24**: {*//'* cat();  
 get\_char();  
  
 bool tag1 = false;  
 if(C == '**\n**') {  
 error(**24**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 if(C == '**\\**') {  
 cat();  
 get\_char();  
 if (C == '**\n**') {  
 tag1 = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 cat();  
 get\_char();  
 if (C == '**\n**') {  
 tag1 = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 }  
 while(C != '**\'**') {  
 cat();  
 get\_char();  
 if(C == '**\n**') {  
 tag1 = true;  
 error(**24**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 if(C == '**\\**') {  
 cat();  
 get\_char();  
 if (C == '**\n**') {  
 tag1 = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 cat();  
 get\_char();  
 if (C == '**\n**') {  
 tag1 = true;  
 error(**23**);  
 lineNum++;  
 state = **0**;  
 break;  
 }  
 }  
 }  
  
 if(tag1) break;  
 cat();  
 state = **0**;  
  
 idNum++;  
 itoa(idNum, idNumStr, **10**);  
 numofMark++;  
 MarkTable &nowMark = mark[numofMark];  
 nowMark.**type** = **idType**;  
  
 strcpy(nowMark.**mark**, "id");  
 strcpy(nowMark.**val**, idNumStr);  
 strcpy(nowMark.**str**, token);  
 retMark(nowMark);  
 break;  
 }  
 case **13**:  
 error(**13**);  
 state = **0**;  
 break;  
 }  
 }while(C != **EOF**);  
 color(**7**);  
 printf("line:%d ",lineNum);  
 printf("idNumber:%d ",idNum);  
 printf("characterNumber:%d ",charNum);  
 printf("errorNum:%d**\n**",errorNum);  
  
 fprintf(dptr, "line:%d ",lineNum);  
 fprintf(dptr, "idNumber:%d ",idNum);  
 fprintf(dptr, "characterNumber:%d ",charNum);  
 fprintf(dptr, "errorNum:%d**\n**",errorNum);  
  
}