# 程序设计 1

**题目：**词法分析程序的设计与实现。

**实验内容：**设计并实现 C 语言的词法分析程序，要求实现如下功能。

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

（2） 可以识别并跳过源程序中的注释。

（3） 可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果。

（4） 检查源程序中存在的词法错误，并报告错误所在的位置。

（5） 对源程序中出现的错误进行适当的恢复，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词法错误。

**实现要求：**分别用以下两种方法实现。

方法 1：采用 C/C++作为实现语言，手工编写词法分析程序。

方法 2：编写 LEX 源程序，利用 LEX 编译程序自动生成词法分析程序。

**实验目的：**用手工方式设计并实现词法分析程序，深刻理解词法分析的主要任务、词法分析程序与语法分析程序之间的关系、词法分析程序的输入输出、单词符号的描述及识别以及整个词法分析过程。

**输入：**C 语言源程序（.cpp）文件

**输出：**由源程序字符串转换成的记号序列，以<记号，属性>的格式输出(如下表所示)

| 字符串 | 记号 | 属性 | 字符串 | 记号 | 属性 |
|---|---|---|---|---|---|
| if | if | - | + | + | - |
| then | then | - | - | - | - |
| id | id | 符号表入口指针 | * | * | - |
| num | num | 常数值 | / | / | - |
| < | < | - | ( | ( | - |

**设计过程：**

1. 语言说明

（1） 标识符：以下划线或字母开头的后跟字母或数字组成的字符串。

（2） 关键字：C 语言中的所有关键字（共 32 个）。

（3） 常数（包括无符号、有符号、十六进制、八进制、二进制）：前二者由整数部分、可选的小数部分和可选的指数部分构成。

（4） 关系运算符：<,<=,>=,>,==,!=,. ,->,!,&&,||, ~

（5） 算术运算符：+,-,*,/,%,++,--, &, ^,|,<<,>>

（6） 标点符号：{,},[,],(,),,,;,:

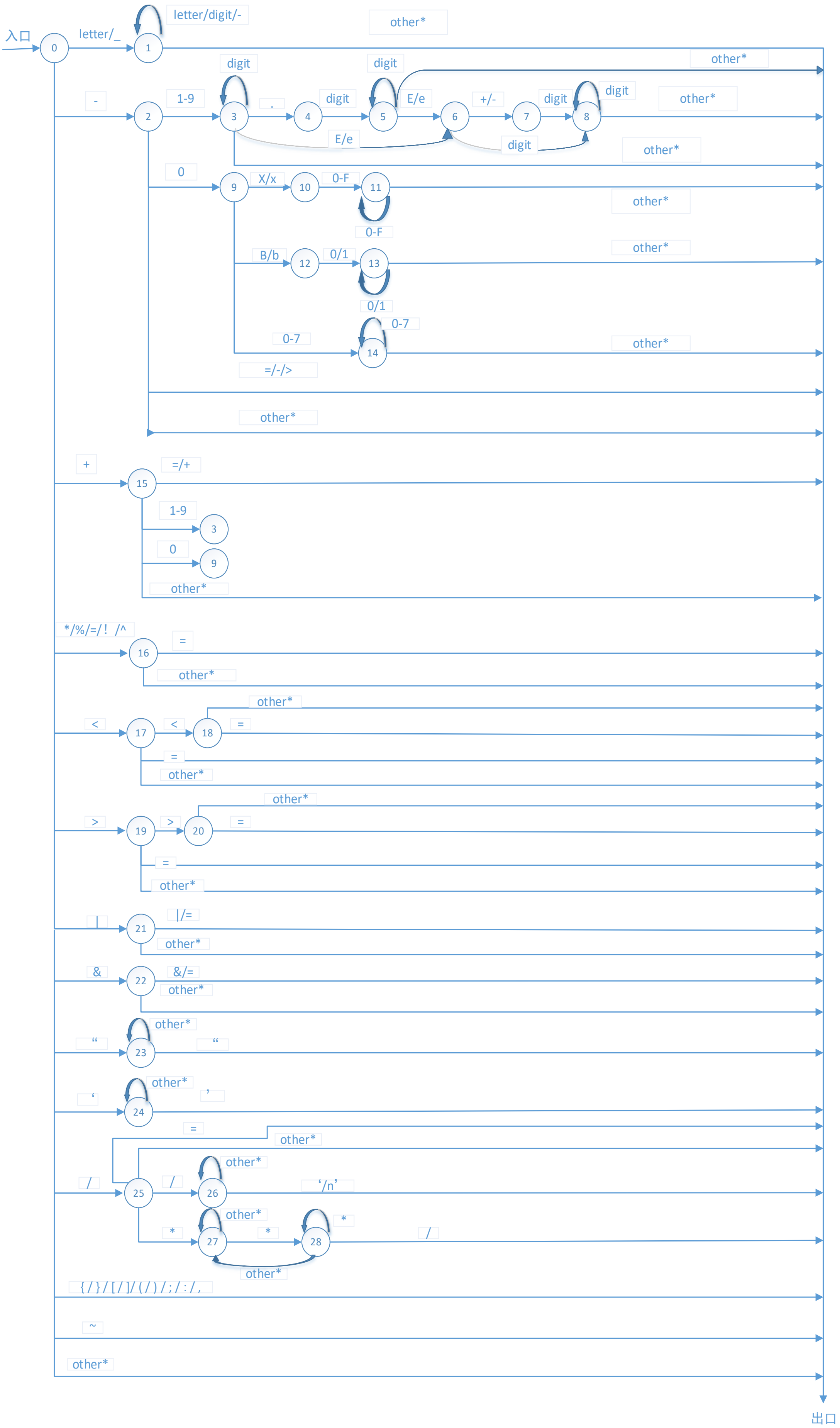
（7） 赋值号：=,+=,-=,*=,/=,&=,^=,|=,<<=,>>=

（8） 注释标记：以”/*”开头以”*/”结束，以及以”//”开头。

（9） 其他： #、空格或\n 和\t 等自动跳过。

2. 设计识别各类字符的状态转换图

根据每种记号的文法构造出相应的状态转换图，让这些状态转换图共用一个初态（如下图所示）。


入口
0
letter/_
1
letter/digit/-
other*
-
2
1-9
3
digit
.
4
digit
5
digit
E/e
6
+/-
7
digit
8
digit
other*
other*
E/e
digit
other*
0
9
X/x
10
0-F
11
0-F
other*
B/b
12
0/1
13
0/1
other*
0-7
14
0-7
other*
=/-/>
other*
+
15
=/+
1-9
3
0
9
other*
*/%/=/! /^
16
=
other*
<
17
<
18
=
other*
=
other*
>
19
>
20
=
other*
=
other*
|
21
|/=
other*
&
22
&/=
other*
“
23
other*
“
‘
24
other*
’
=
other*
/
25
/
26
other*
‘/n’
*
27
other*
*
28
*
/
other*
{/}/[/]/(/)/;/:/,
~
other*
出口

## 3. 源代码

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <cstring>
#define KEY         0  //关键字
#define ID          1  //标识符
#define RELOP       2  //运算符
#define NUM         3  //常数
#define STRING      4  //字符串
#define PUNCTUATION 5  //标点符号
#define ANNOTATION  6  //注释
#define BUF_LENGTH  64 //缓冲区长度
#define L_END       31 //左半边终止位
#define R_END       63 //右半边终止位
#define START       0  //开始指针
using namespace std;

string file;                    //文件名
ifstream in;                    //文件指针
char C;                         //存放当前读入的字符
int state = 0;                  //状态
int linenum, wordnum, charnum;  //行数，单词数，字符数
char buffer[BUF_LENGTH];        //缓冲区字符数组
int l_end, r_end;               //左右半区终点
int _forward = 0;               //前进指针
bool lflag, rflag;              //是否需要填充缓冲区标记
string token;                   //存放当前正在识别的单词字符串
string char_to_string(char c);  //char 型转换为 string 型
string int_to_string(int c);    //int 型转换为 string 型
vector<string> id;              //自己定义的标识符表
vector<string> keyword;         //关键字
vector<string> num;             //常数表
vector<string> literal;         //""以及''中的字符串
void get_char();                //从输入缓冲区中读入一个字符放入变量 C 中
void get_nbc();                 //检查 C 中是否有空格
bool letter(char c);            //判断 C 中的字符是否为字母，是则返回 true 否则返回 false
bool digit(char c);             //判断 C 中的字符食肉为数字，若是返回 true 否则返回 false
void retract();                 //向前指针退回一位
void init_key();                //生成关键字表
int iskey(string token);        //判断标识符是否在关键字表里
void init();                    //初始化所有参数
void fillBuffer(int i);         //填充缓冲区，0：填充左边 1：填充右边
void work();                    //词法分析主体部分
void output(int type, string out);//以<记号,属性>的形式输出分析结果
void error();                     //错误处理程序
void startwork(string f);         //读取文件开始分析
vector<string> key_table;         //关键字表

void init_key() {
    key_table.clear();
    key_table.push_back("char");    key_table.push_back("double");
key_table.push_back("enum");    key_table.push_back("float");
    key_table.push_back("int");     key_table.push_back("long");
key_table.push_back("short");   key_table.push_back("signed");
    key_table.push_back("struct");  key_table.push_back("union");
key_table.push_back("unsigned"); key_table.push_back("void");
```

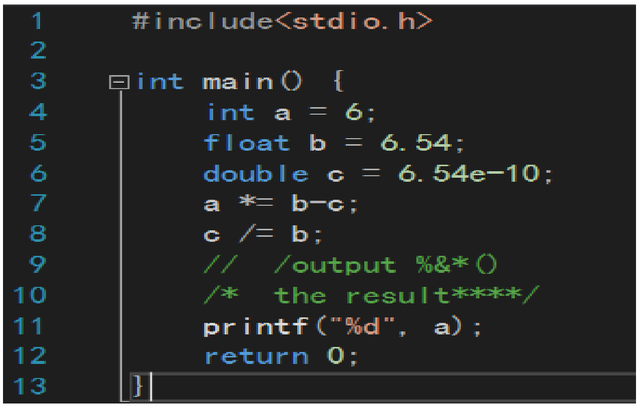
```
    key_table.push_back("for");      key_table.push_back("do");
key_table.push_back("while");    key_table.push_back("break");
    key_table.push_back("continue"); key_table.push_back("if");
key_table.push_back("else");     key_table.push_back("goto");
    key_table.push_back("switch");   key_table.push_back("case");
key_table.push_back("default");  key_table.push_back("return");
    key_table.push_back("auto");     key_table.push_back("extern");
key_table.push_back("register"); key_table.push_back("static");
    key_table.push_back("const");    key_table.push_back("sizeof");
key_table.push_back("typedef");  key_table.push_back("volatile");
}
bool letter(char c) {
    return (c >= 'a'&&c <= 'z' || c >= 'A'&&c <= 'Z');

}
bool digit(char c) {
    return (c >= '0'&&c <= '9');
}
int iskey(string token) {
    vector<string>::iterator it;
    for (it = key_table.begin(); it != key_table.end(); it++) {
        if (token == (*it))
            return it - key_table.begin();
    }
    return -1;
}
string char_to_string(char c) {
    string str;
    stringstream stream;
    stream << c;
    str = stream.str();
    return str;
}
string int_to_string(int c) {
    string str;
    stringstream stream;
    stream << c;
    str = stream.str();
    return str;
}
void init() {
    id.clear();
    l_end = L_END;
    r_end = R_END;
    _forward = 0;
    lflag = rflag = false;
    buffer[l_end] = buffer[r_end] = EOF;
    fillBuffer(0);
    linenum = wordnum = charnum = 0;
}
void fillBuffer(int i) {
    if (i == 0) {
        if (lflag == false) {
            in.read(buffer, l_end);
            if (in.gcount() != l_end) {
                buffer[in.gcount()] = EOF;
            }
        }
        else {
            lflag = false;
        }
    }
    else {
        if (rflag == false) {
            in.read(buffer + l_end + 1, l_end);
            if (in.gcount() != l_end) {
                buffer[in.gcount() + l_end + 1] = EOF;
            }
        }
        else {
            rflag = false;
        }
    }
}
void get_char() {
    C = buffer[_forward];
    if (C == EOF)
        return;
    if (C == '\n') {
        linenum++;
        charnum++;
    }
    else {
        charnum++;
    }
    _forward++;
    if (buffer[_forward] == EOF) {
        if (_forward == l_end) {
            fillBuffer(1);
            _forward++;
        }
        else if (_forward == r_end) {
            fillBuffer(0);
            _forward = START;
        }
    }
}
void get_nbc() {
    while (C == ' ' || C == '\t' || C == '\n')
        get_char();
}
void retract() {
    if (_forward == 0) {
        lflag = true;
        _forward = r_end - 1;
    }
    else {
        _forward--;
        if (_forward == l_end) {
            rflag = true;
            _forward--;
        }
    }
}
void output(int type, string out) {
    switch (type) {
    case KEY:
        keyword.push_back(out);
        cout << "<" << out << ", >" << endl;
        break;
    case ID:
        cout << "<id," << atoi(out.c_str()) << ">" << endl;
        break;
    case NUM:
        num.push_back(out);
        cout << "<num," << out << ">" << endl;
        break;
    case RELOP:
        cout << "<" << out << ", >" << endl;
        break;
    case STRING:
        literal.push_back(out);
        cout << "<string," << out << ">" << endl;
        break;
    case PUNCTUATION:
        cout << "<" << out << ", >" << endl;
        break;
    case ANNOTATION:
        cout << "<annotation," << out << ">" << endl;
        break;
    default:
        break;
    }
    wordnum++;
}
void error() {
    cout << "Line:" << linenum + 1 << " error!" << endl;
}
void work() {
    do {
        switch (state) {
        case 0:
            token.clear();
            get_char();
            get_nbc();
            if (C == '_' || letter(C)) {
                state = 1;
            }
            else if (C == '-') {
                state = 2;
            }
            else if (C >= '1'&&C <= '9') {
                state = 3;
            }
            else if (C == '0') {
                state = 9;
            }
            else if (C == '+') {
                state = 15;
            }
            else if (C == '*' || C == '%' || C == '!' || C == '=' || C == '^') {
                state = 16;
            }
            else if (C == '~' || C == '.') {
                output(RELOP, char_to_string(C));
                state = 0;
            }
            else if (C == '{' || C == '}' || C == '[' || C == ']' || C == '(' || C == ')' || C == ';' || C == ':' || C == ',') {
                output(PUNCTUATION, char_to_string(C));
                state = 0;
            }
            else if (C == '<') {
                state = 17;
            }
            else if (C == '>') {
                state = 19;
            }
            else if (C == '|') {
                state = 21;
            }
            else if (C == '&') {
                state = 22;
            }
            else if (C == '"') {
                state = 23;
            }
            else if (C == '\'') {
                state = 24;
            }
            else if (C == '/') {
                state = 25;
            }
            else if (C == '#') {
                state = 29;
            }
            else
                state = 30;
            break;
        case 1:
            token.push_back(C);
            get_char();
            if (letter(C) || digit(C) || C == '_') {
                state = 1;
            }
            else {
                retract();
                state = 0;
                if (iskey(token) != -1) {
                    output(KEY, key_table[iskey(token)]);//直接输出关键字
                }
                else {
                    id.push_back(token);
                    int locate = id.size() - 1;
                    output(ID, int_to_string(locate));
                    state = 0;
                }
            }
            break;
        case 2:
            token.push_back(C);
            get_char();
            if (C >= '1'&&C <= '9') {
                state = 3;
            }
            else if (C == '0') {
                state = 9;
            }
            else  if (C == '.') {
                state = 4;
            }
            else if (C == '=' || C == '-' || C == '>') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 3:
            token.push_back(C);
            get_char();
            if (digit(C)) {
                state = 3;
            }
            else if (C == '.') {
                state = 4;
            }
            else if (C == 'E' || C == 'e') {
                state = 6;
            }
            else {
                retract();
                output(NUM, token);
                state = 0;
            }
            break;
        case 4:
            token.push_back(C);
            get_char();
            if (digit(C)) {
                state = 5;
            }
            else {
                error();
                state = 0;
            }
            break;
        case 5:
            token.push_back(C);
            get_char();
            if (digit(C)) {
                state = 5;
            }
            else if (C == 'E' || C == 'e') {
                state = 6;
            }
            else {
                retract(); output(NUM, token);
                state = 0;
            }
            break;
```

```
case 6:
    token.push_back(C);
    get_char();
    if (C == '+' || C == '-') {
        state = 7;
    }
    else if (digit(C)) {
        state = 8;
    }
    else {
        retract();
        error();
        state = 0;
    }
    break;
case 7:
    token.push_back(C);
    get_char();
    if (digit(C)) {
        state = 8;
    }
    else {
        retract();
        error();
        state = 0;
    }
    break;
case 8:
    token.push_back(C);
    get_char();
    if (digit(C)) {
        state = 8;
    }
    else {
        retract();
        output(NUM, token);
        state = 0;
    }
    break;
case 9:
    token.push_back(C);
    get_char();
    if (C == '.') {
        state = 4;
    }
    else if (C == 'X' || C == 'x') {
        state = 10;
    }
    else if (C == 'B' || C == 'b') {
        state = 12;
    }
    else if (C >= '0'&&C <= '7') {
        state = 14;
    }
    else {
        retract();
        output(NUM, token);
        state = 0;
    }
```

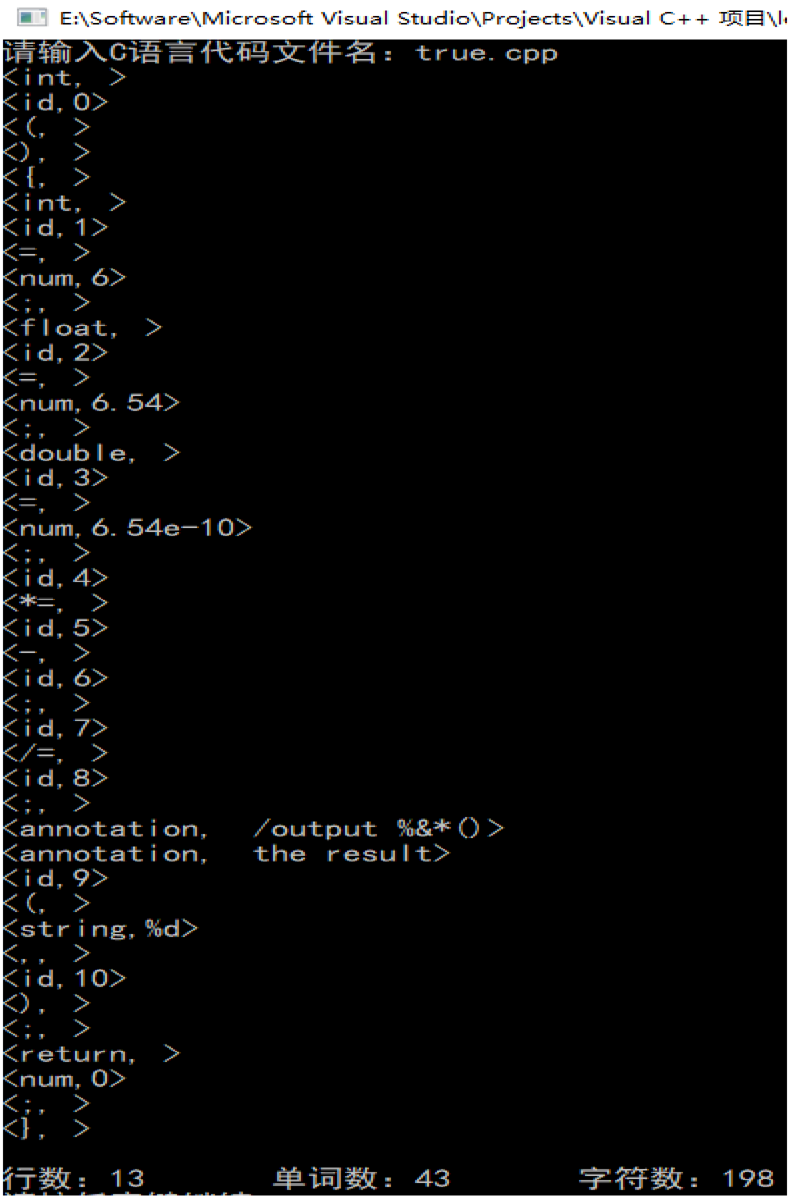
```
            break;
        case 10:
            token.push_back(C);
            get_char();
            if ((C >= '0'&&C <= '9') || (C >= 'A'&&C <= 'F') || (C >= 'a'&&C <= 'f')) {
                state = 11;
            }
            else {
                retract();
                error();
                state = 0;
            }
            break;
        case 11:
            token.push_back(C);
            get_char();
            if ((C >= '0'&&C <= '9') || (C >= 'A'&&C <= 'F') || (C >= 'a'&&C <= 'f')) {
                state = 11;
            }
            else {
                retract();
                output(NUM, token);
                state = 0;
            }
            break;
        case 12:
            token.push_back(C);
            get_char();
            if (C == '0' || C == '1') {
                state = 13;
            }
            else {
                retract();
                error();
                state = 0;
            }
            break;
        case 13:
            token.push_back(C);
            get_char();
            if (C == '0' || C == '1') {
                state = 13;
            }
            else {
                retract();
                output(NUM, token);
                state = 0;
            }
            break;
        case 14:
            token.push_back(C);
            get_char();
            if (C >= '0'&&C <= '7') {
                state = 14;
            }
            else {
                retract();
                output(NUM, token);
                state = 0;
            }
            break;
        case 15:
            token.push_back(C);
            get_char();
            if (C == '=' || C == '+') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else if (C >= '1'&&C <= '9') {
                state = 3;
            }
            else if (C == '0') {
                state = 9;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 16:
            token.push_back(C);
            get_char();
            if (C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 17:
            token.push_back(C);
            get_char();
            if (C == '<') {
                state = 18;
            }
            else if (C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 18:
            token.push_back(C);
            get_char();
            if (C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 19:
            token.push_back(C);
            get_char();
            if (C == '>') {
                state = 20;
            }
            else if (C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 20:
            token.push_back(C);
            get_char();
            if (C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 21:
            token.push_back(C);
            get_char();
            if (C == '|' || C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 22:
            token.push_back(C);
            get_char();
            if (C == '&' || C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 23:
            get_char();
            if (C == '"') {
                output(STRING, token);
                state = 0;
            }
            else {
                token.push_back(C);
                state = 23;
            }
            break;
        case 24:
            get_char();
            if (C == '\'') {
                output(STRING, token);
                state = 0;
            }
            else {
                token.push_back(C);
                state = 24;
            }
            break;
        case 25:
            token.push_back(C);
            get_char();
            if (C == '/') {
                state = 26;
            }
            else if (C == '*') {
                state = 27;
            }
            else if (C == '=') {
                token.push_back(C);
                output(RELOP, token);
                state = 0;
            }
            else {
                retract();
                output(RELOP, token);
                state = 0;
            }
            break;
        case 26:
            get_char();
            if (C == '\n') {
                output(ANNOTATION, token.substr(1,token.size()-1));
                state = 0;
            }
            else {
                token.push_back(C);
                state = 26;
```

```
            }
            break;
        case 27:
            get_char();
            if (C == '*') {
                state = 28;
            }
            else {
                token.push_back(C);
                state = 27;
            }
            break;
        case 28:
            get_char();
            if (C == '*') {
                state = 28;
            }
            else if (C == '/') {
                output(ANNOTATION, token.substr(1,token.size()-1));
                state = 0;
            }
            else {
                token.push_back('*');
                token.push_back(C);
                state = 27;
            }
            break;
        case 29:
            while (C != '\n') {
                get_char();
            }
            state = 0;
            break;
        case 30:
            cout << "Line: " << linenum+1 << " error!" << endl;
            state = 0;
            break;
        }
    } while (C != EOF);
}
void startwork(string f) {
    char firename[20];
    file = f;
    strcpy(firename, file.c_str());
    in.open(firename);
    if (in) {
        init();
        work();
        cout << endl;
        cout << "行数：" << linenum << "       ";
        cout << "单词数：" << wordnum << "       ";
        cout << "字符数：" << charnum << "       " << endl;
    }
    else {
        cout << "文件打开失败！" << endl;
    }
}
void main() {
    init_key();
    string filename;
    cout << "请输入C语言代码文件名：";
    cin >> filename;
    startwork(filename);
    system("pause");
}
```

**实验结果：**输入正确的程序 true.cpp



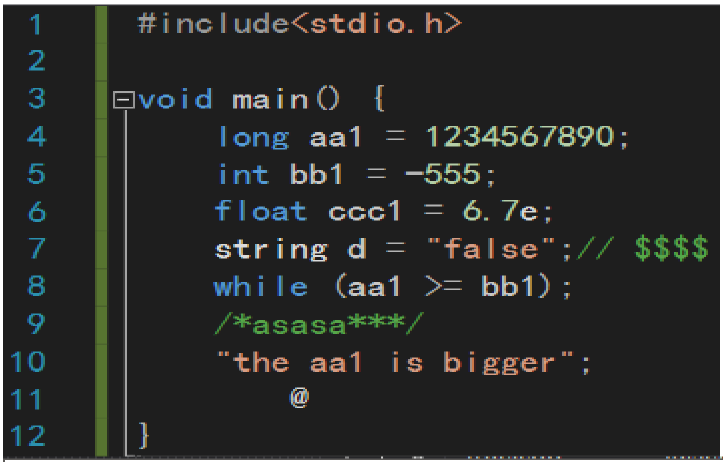
```
#include<stdio.h>

int main() {
    int a = 6;
    float b = 6.54;
    double c = 6.54e-10;
    a *= b-c;
    c /= b;
    //  /output %&*()
    /*  the result****/
    printf("%d", a);
    return 0;
}
```



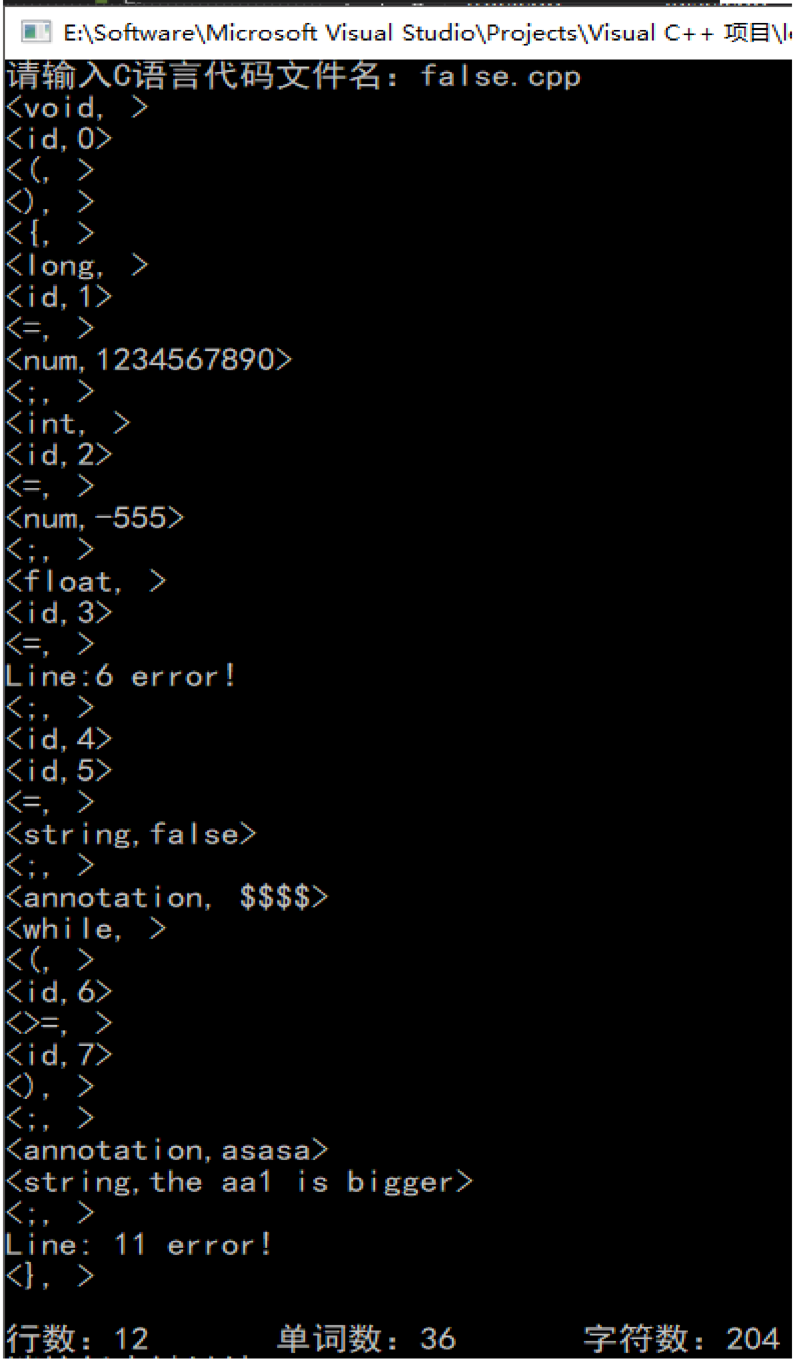
E:\Software\Microsoft Visual Studio\Projects\Visual C++ 项目\l

```
请输入C语言代码文件名：true.cpp
<int, >
<id,0>
<(, >
<), >
<{, >
<int, >
<id,1>
<=, >
<num,6>
<;, >
<float, >
<id,2>
<=, >
<num,6.54>
<;, >
<double, >
<id,3>
<=, >
<num,6.54e-10>
<;, >
<id,4>
<*=, >
<id,5>
<-, >
<id,6>
<;, >
<id,7>
</=, >
<id,8>
<;, >
<annotation,  /output %&*()>
<annotation,  the result>
<id,9>
<(, >
<string,%d>
<,, >
<id,10>
<), >
<;, >
<return, >
<num,0>
<;, >
<}, >

行数：13          单词数：43          字符数：198
```

输入错误的程序 false.cpp，其代码段如下图：



```
1   #include<stdio.h>
2
3  ⊟void main() {
4         long aa1 = 1234567890;
5         int bb1 = -555;
6         float ccc1 = 6.7e;
7         string d = "false";// $$$$
8         while (aa1 >= bb1);
9         /*asasa***/
10        "the aa1 is bigger";
11            @
12   }
```



E:\Software\Microsoft Visual Studio\Projects\Visual C++ 项目\l

```
请输入C语言代码文件名：false.cpp
<void, >
<id,0>
<(, >
<), >
<{, >
<long, >
<id,1>
<=, >
<num,1234567890>
<;, >
<int, >
<id,2>
<=, >
<num,-555>
<;, >
<float, >
<id,3>
<=, >
Line:6 error!
<;, >
<id,4>
<id,5>
<=, >
<string,false>
<;, >
<annotation, $$$$>
<while, >
<(, >
<id,6>
<>=, >
<id,7>
<), >
<;, >
<annotation,asasa>
<string,the aa1 is bigger>
<;, >
Line: 11 error!
<}, >

行数：12        单词数：36        字符数：204
```