# 实验报告

曹桢 学号：2014211182 班级：2014211302

## 实验题目：

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：

采用C/C++作为实现语言，手工编写词法分析程序。

## 程序设计说明

1. 输入文件：C语言源程序
2. 语言说明：

标识符：以字母开头的、后跟字母或数字组成的符号串。

保留字：C语言的32个保留字。

无符号数：同C语言中的无符号数。

关系运算符：<、<=、!=、==、>=、>。

标点符号：+、-、\*、/、(、)、:、′、；等。

赋值号： =

注释标记：以‘/\*’开始，以‘\*/’结束。或者以“//”开头的一行

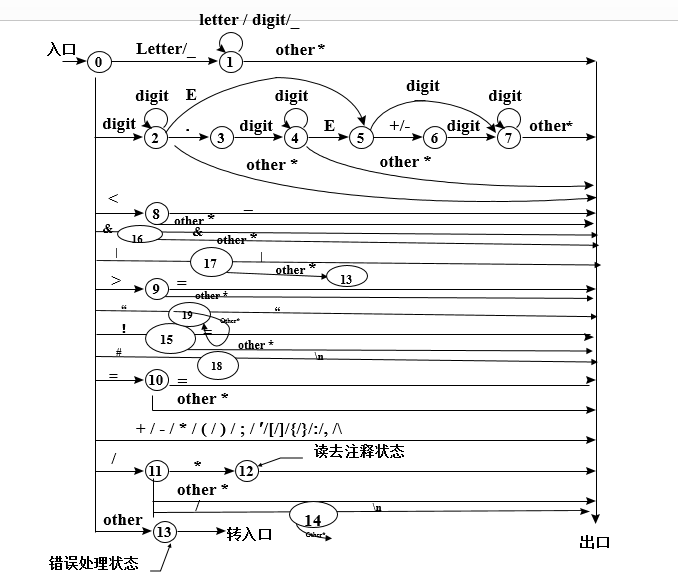
单词符号间的分隔符：空格

1. 输出：

翻译表：

|  |  |  |
| --- | --- | --- |
| 正规表达式 | 记号 | 属性 |
| Reserve | Reserve | - |
| Id | id | 符号表入口指针 |
| Num | NUM | 常数值 |
| < | relop | LT |
| <= | Relop | LE |
| == | Relop | EQ |
| != | Relop | NE |
| > | Relop | GT |
| >= | relop | GE |
| = | Assign\_op | - |
| + | + | - |
| - | - | - |
| \* | \* | - |
| / | / | - |
| ( | ( | - |
| ) | ) | - |
| ‘ | ‘ | - |
| ; | ; | - |
| “…” | string | … |
| ! | Logic\_op | NOT |
| && | Logic\_op | AND |
| || | Logic\_op | OR |
| [ | [ | - |
| ] | ] | - |
| { | { | - |
| } | } | - |
| : | : | - |
| , | , | - |
| “ | “ | - |
| \ | \ | - |
| & | & | - |
| # | # | - |

1. 状态转换图



1. 高层数据结构设计
2. 类型定义

struct ID{ //用户自定义标识符

string name; //标识符名称

int pos; //标识符入口地址

};

struct Error{ //错误

int where; //错误行数

string err; //错误的字符串

int kind; //错误种类

};

1. 常量定义

const int bufferSize = 1000; //半个缓冲区的大小

1. 关键变量定义

int state; //自动机状态

char C; //当前读入的字符

string token; //当前正在识别的单词字符串

char buffer1[bufferSize]; //输入缓冲区左半区

char buffer2[bufferSize]; //输入缓冲区右半区 char \*forward; //向前指针

char \*lexemebegin; //指向buffer中当前单词的开始位置

int iskey; //值为-1，表示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

int line; //统计行数

int charactor; //统计字符总数

int keychar; //统计关键字个数

int relop; //统计关系运算符数

int assign\_op; //统计赋值运算符数

int logic\_op; //统计逻辑运算符数

int numcount; //统计常数个数

int otherchar; //统计其他字符数

int idcount; //统计用户标识符数

int stringcount; //统计“”字符串数

vector <ID> table; //标识符表

vector <Error> countError; //错误信息表

fstream file; //源程序

1. 函数说明

//根据forward的指示从buffer中读一个字符，并把它放入变量C中，然后，移动forward，使之指向下一个字符。

void get\_char();

//检查C中的字符是否为空格，若是，则反复调  
 用过程get\_char，直到C中进入一个非空字符为止。

void get\_nbc();

//把C中的字符连接在token中的字符串后面

void cat();

//判断C中的字符是否为字母

bool letter();

//判断C中的字符是否为数字

bool digit();

//向前指针forward后退一个字符

bool retract();

//根据token中的单词查关键字表

int reserve();

//将识别出来的标识符（即token中的单  
 词）插入符号表，返回该单词在符号表中的位置指针

int table\_insert();

//对发现的错误进行相应的处理

void error(int which);

//自动机

void automachine();

## 源代码

#include<iostream>

#include<fstream>

#include<cstring>

#include<cstdio>

#include <cstdlib>

#include<vector>

using namespace std;

const int bufferSize = 60; //半个缓冲区的大小

struct ID{ //用户自定义标识符

string name; //标识符名称

int pos; //标识符入口地址

};

struct Error{ //错误

int where; //错误行数

string err; //错误的字符串

int kind; //错误种类

};

int state; //自动机状态

char C; //当前读入的字符

string token; //当前正在识别的单词字符串

char buffer1[bufferSize]; //输入缓冲区左半区

char buffer2[bufferSize]; //输入缓冲区右半区

char \*forward; //向前指针

char \*lexemebegin; //指向buffer中当前单词的开始位置

int iskey; //值为-1，表示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

int line; //统计行数

int charactor; //统计字符总数

int keychar; //统计关键字个数

int relop; //统计关系运算符个数

int assign\_op; //统计赋值运算符个数

int logic\_op; //统计逻辑运算符个数

int numcount; //统计常数个数

int otherchar; //统计其他字符数

int idcount; //统计用户标识符数

int stringcount; //统计“”字符串数

vector <ID> table; //标识符表

vector <Error> countError; //错误信息表

fstream file; //源程序

//根据forward的指示从buffer中读一个字符，并把它放入变量C中，然后，移动forward，使之指向下一个字符。

void get\_char();

//检查C中的字符是否为空格，若是，则反复调用过程get\_char，直到C中进入一个非空字符为止。

void get\_nbc();

//把C中的字符连接在token中的字符串后面

void cat();

//判断C中的字符是否为字母

bool letter();

//判断C中的字符是否为数字

bool digit();

//向前指针forward后退一个字符

bool retract();

//根据token中的单词查关键字表

int reserve();

//将识别出来的标识符（即token中的单词）插入符号表，返回该单词在符号表中的位置指针

int table\_insert();

//对发现的错误进行相应的处理

void error(int which);

//自动机

void automachine();

int main()

{

line = 1;

charactor = 0;

keychar = 0;

relop = 0;

assign\_op = 0;

logic\_op = 0;

numcount = 0;

otherchar = 0;

idcount = 0;

stringcount = 0;

state = 0;

C = ' ';

token = "";

buffer1[bufferSize-1] = -1;

buffer2[bufferSize-1] = -1;

lexemebegin = buffer1;

forward = buffer1;

char filename[100] = "test1.txt";

file.open(filename, ios::in);

if(!file){

cout<<"Failed to open "<<filename<<"!"<<endl;

return -1;

}

int i = 0;

while(i < bufferSize-1){

buffer1[i]=file.get();

i++;

}

do

{

automachine();

}while(C!=EOF);

cout<<"-----------------------------\n- Statistics:"<<endl;

cout<<"totoal line: "<<line<<endl;

cout<<"total charactor:"<<charactor<<endl;

cout<<"key charactor:"<<keychar<<endl;

cout<<"relation operator: "<<relop<<endl;

cout<<"assign operator:"<<assign\_op<<endl;

cout<<"logic operator:"<<logic\_op<<endl;

cout<<"number: "<<numcount<<endl;

cout<<"other charactor:"<<otherchar<<endl;

cout<<"how many ids set in code:"<<idcount<<endl;

cout<<"how many ids set in table:"<<table.size()<<endl;

cout<<"how many strings: "<<stringcount<<endl;

if(countError.empty()){

cout<<"-----------------------------\n- Errors: 0"<<endl;

}

else{

cout<<"-----------------------------\n- Errors:"<<endl;

cout<<"line"<<'\t'<<"Error"<<endl;

for(int i = 0; i < countError.size(); i++){

switch(countError[i].kind){

case 0:

cout<<countError[i].where<<'\t'<<"[Error]"<<countError[i].err<<" There should be digits after . to compose a number."<<endl;

break;

case 1:

cout<<countError[i].where<<'\t'<<"[Error]"<<countError[i].err<<" There should be digits or + or - after E to compose a number."<<endl;

break;

case 2:

cout<<countError[i].where<<'\t'<<"[Error]"<<countError[i].err<<" There should be digits after + or - to compose a number."<<endl;

break;

case 3:

cout<<countError[i].where<<'\t'<<"[Error]"<<countError[i].err<<" Invalid character!"<<endl;

break;

default:

cout<<countError[i].where<<'\t'<<"[Error]"<<"Unknown Error!"<<endl;

}

}

}

file.close();

return 0;

}

void get\_char()

{

if((\*forward) == -1){

if(forward== buffer1+bufferSize-1){

int i = 0;

while(i < bufferSize-1){

buffer2[i]=file.get();

i++;

}

forward=buffer2;

}

else if(forward== buffer2+bufferSize-1){

int i = 0;

while( i < bufferSize-1){

buffer1[i]=file.get();

i++;

}

forward = buffer1;

}

else

{

file.close();

}

}

C = \*forward;

forward++;

}

void get\_nbc()

{

while(C == ' ' || C == '\t' || C == '\n'){

if(C == '\n'){

line++;

}

get\_char();

}

}

void cat()

{

token = token + C;

}

bool letter()

{

if(C >= 'A' && C <= 'z') return true;

else return false;

}

bool digit()

{

if(C >= '0' && C <= '9') return true;

else return false;

}

bool retract()

{

if(forward == buffer1){

forward = buffer2+bufferSize-1;

}

else if(forward == buffer2){

forward = buffer1+bufferSize-1;

}

else{

forward--;

}

}

int reserve()

{

if( token == "void" || token == "char" || token == "int" || token == "float"

|| token == "double" || token == "short" || token == "long" || token == "signed"

|| token == "unsigned" || token == "struct" || token == "union" || token == "enum"

|| token == "typedef" || token == "sizeof" || token == "auto" || token == "static"

|| token == "register" || token == "extern" || token == "const" || token == "volatile"

|| token == "return" || token == "continue" || token == "break" || token == "goto"

|| token == "if" || token == "else" || token == "switch" || token == "case"

|| token == "default" || token == "for" || token == "do" || token == "while"){

return 1;

}

else return -1;

}

int table\_insert()

{

int len = table.size();

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

if(table[i].name == token){

return i;

}

}

ID newid;

newid.name = token;

newid.pos = len;

table.push\_back(newid);

return len;

}

void error(int which)

{

Error e;

e.where = line;

e.err = token;

e.kind = which;

countError.push\_back(e);

lexemebegin = forward;

}

void automachine()

{

switch ( state ) {

case 0:

get\_char();

get\_nbc();

if(letter() || C == '\_'){

state = 1;

}

else if(digit()){

state = 2;

}

else{

switch ( C ) {

case '<': state = 8; break;

case '>': state = 9; break;

case '!': state = 15; break;

case '&': state = 16; break;

case '|': state = 17; break;

case '#': state = 18; break;

case '/': state = 11; break;

case '=': state = 10; break;

case '+': state = 0; cout << '+' << '\t' << '(' << '+' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '-': state = 0; cout << '-' << '\t' << '(' << '-' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '\*': state = 0; cout << '\*' << '\t' << '(' << '\*' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '(': state = 0; cout << '(' << '\t' << '(' << '(' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case ')': state = 0; cout << ')' << '\t' << '(' << ')' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '[': state = 0; cout << '[' << '\t' << '(' << '[' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case ']': state = 0; cout << ']' << '\t' << '(' << ']' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '{': state = 0; cout << '{' << '\t' << '(' << '{' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '}': state = 0; cout << '}' << '\t' << '(' << '}' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case ';': state = 0; cout << ';' << '\t' << '(' << ';' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case ':': state = 0; cout << ':' << '\t' << '(' << ':' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case ',': state = 0; cout << ',' << '\t' << '(' << ',' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '\'': state = 0; cout << '\'' << '\t' << '(' << '\'' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '"': state = 19; cout << '"' << '\t' << '(' << '"' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

case '\\': state = 0; cout << '\\' << '\t' << '(' << '\\' << ',' << '-' << ')' << endl; charactor++; otherchar++; lexemebegin = forward; break;

default: state = 13; break;

};

}

break;

case 1:

cat();

get\_char();

if ( letter() || digit() || C == '\_') state = 1;

else {

retract();

state = 0;

iskey = reserve();

if ( iskey == 1 ) {

cout << token << '\t' << '(' << token << ',' << '-' << ')' << endl;

keychar++;

charactor++;

token.clear();

lexemebegin = forward;

}

else {

int identry = table\_insert();

cout << token << '\t' << '(' << "ID" << ',' << identry << ')' << endl;

idcount++;

charactor++;

lexemebegin = forward;

token.clear();

};

};

break;

case 2:

cat();

get\_char();

if(digit()){

state = 2;

}

else {

switch ( C ) {

case '.' : state=3; break;

case 'E': state=5; break;

default:

retract();

state = 0;

cout << token << '\t'<< '(' << "NUM" << ',' << atoi(token.c\_str()) << ')' << endl;

numcount++;

charactor++;

lexemebegin = forward;

token.clear();

break;

};

}

break;

case 3:

cat();

get\_char();

if(digit()){

state = 4;

}

else{

retract();

error(0);

state = 0;

token.clear();

}

break;

case 4:

cat();

get\_char();

if(digit()){

state = 4;

}

else if(C == 'E'){

state = 5;

}

else{

retract();

state=0;

cout << token << '\t'<< '(' << "NUM" << ',' << atof(token.c\_str()) << ')' << endl;

numcount++;

charactor++;

lexemebegin = forward;

token.clear();

}

break;

case 5:

cat();

get\_char();

if(digit()){

state = 7;

}

else if(C == '+' || C == '-'){

state = 6;

}

else{

retract();

error(1);

state = 0;

token.clear();

}

break;

case 6:

cat();

get\_char();

if(digit()){

state = 7;

}

else{

retract();

error(2);

state = 0;

token.clear();

}

break;

case 7:

cat();

get\_char();

if(digit()){

state = 7;

}

else{

retract();

state = 0;

cout << token << '\t'<< '(' << "NUM" << ',' << atof(token.c\_str()) << ')' << endl;

numcount++;

charactor++;

lexemebegin = forward;

token.clear();

}

break;

case 8:

cat();

get\_char();

if(C == '=') {

state = 0;

cout << "<=" << '\t' << '(' << "relop" << ',' << "LE" << ')' << endl;

relop++;

charactor++;

lexemebegin = forward;

token.clear();

}

else {

retract();

state = 0;

cout << '<' << '\t' << '(' << "relop" << ',' << "LT" << ')' << endl;

relop++;

charactor++;

lexemebegin = forward;

token.clear();

}

break;

case 9:

cat();

get\_char();

if(C == '=') {

state = 0;

cout << ">=" << '\t' << '(' << "relop" << ',' << "GE" << ')' << endl;

relop++;

charactor++;

lexemebegin = forward;

token.clear();

}

else {

retract();

state = 0;

cout << '>' << '\t' << '(' << "relop" << ',' << "GT" << ')' << endl;

relop++;

charactor++;

lexemebegin = forward;

token.clear();

}

break;

case 10:

cat();

get\_char();

if(C == '=') {//判断相等

state = 0;

cout << "==" << '\t' << '(' << "relop" << ',' << "EQ" << ')' << endl;

relop++;

charactor++;

lexemebegin = forward;

token.clear();

}

else {//赋值

retract();

state = 0;

cout << '=' << '\t' << '(' << "assign\_op" << ',' << '-' << ')' << endl;

assign\_op++;

charactor++;

lexemebegin = forward;

token.clear();

}

break;

case 11:

cat();

get\_char();

if(C == '\*') {

state = 12;

}

else if(C == '/') {

state = 14;

}

else {

retract();

state = 0;

cout << '/' << '\t' << '(' << '/' << ',' << '-' << ')' << endl;

otherchar++;

charactor++;

lexemebegin = forward;

token.clear();

}

break;

case 12:

get\_char();

token.clear();

while(C != '\*') get\_char();

get\_char();

if(C == '/'){

state = 0;

}

else state = 12;

break;

case 13:

error(3);

state = 0;

break;

case 14:

get\_char();

while(C != '\n') get\_char();

line++;

state = 0;

break;

case 15:

cat();

get\_char();

if(C == '=') {//判断不等

state = 0;

cout << "!=" << '\t' << '(' << "relop" << ',' << "NE" << ')' << endl;

relop++;

charactor++;

lexemebegin = forward;

token.clear();

}

else{

retract();

state = 0;

cout << '!' << '\t' << '(' << "logic\_op" << ',' << "NOT" << ')' << endl;

logic\_op++;

charactor++;

lexemebegin = forward;

token.clear();

}

case 16:

cat();

get\_char();

if(C == '&'){

state = 0;

cout << "&&" << '\t' << '(' << "logic\_op" << ',' << "AND" << ')' <<endl;

logic\_op++;

charactor++;

lexemebegin = forward;

token.clear();

}

else{

retract();

cout << "&" << '\t' << '(' << "&" << ',' << "-" << ')' <<endl;

state = 0;

otherchar++;

charactor++;

lexemebegin = forward;

token.clear();

}

break;

case 17:

cat();

get\_char();

if(C == '|'){

state = 0;

cout << "||" << '\t' << '(' << "logic\_op" << ',' << "OR" << ')' <<endl;

logic\_op++;

charactor++;

lexemebegin = forward;

token.clear();

}

else{

retract();

error(3);

state = 0;

}

break;

case 18:

get\_char();

while(C != '\n') get\_char();

line++;

state = 0;

break;

case 19:

get\_char();

while(C != '"') {

cat();

get\_char();

}

cout << token << '\t' << '(' << "const string" << ',' << token << ')' <<endl;

stringcount++;

charactor++;

cout << '"' << '\t' << '(' << '"' << ',' << '-' << ')' << endl;

otherchar++;

charactor++;

lexemebegin = forward;

token.clear();

state = 0;

break;

}

}

## 测试报告

1、

输入：

#include<stdio.h>

int main()

{

char a;

short b;

int c;

long d;

long long e;

float f;

double g;

printf("Please input seven parts:\n");

scanf("%c%hd%d%ld%lld%f%lf",&a,&b,&c,&d,&e,&f,&g);

printf("The result is:\n");

printf("The 'char' variable is %c,it takes %d byte.\n",a,sizeof(a));

printf("The'short' variable is %hd,it takes %d bytes.\n",b,sizeof(b));

printf("The 'int' variable is %d,it takes %d bytes.\n",c,sizeof(c));

printf("The 'long' variable is %ld,it takes %d bytes.\n",d,sizeof(d));

printf("The 'long long' variable is %lld,it takes %d bytes.\n",e,sizeof(e));

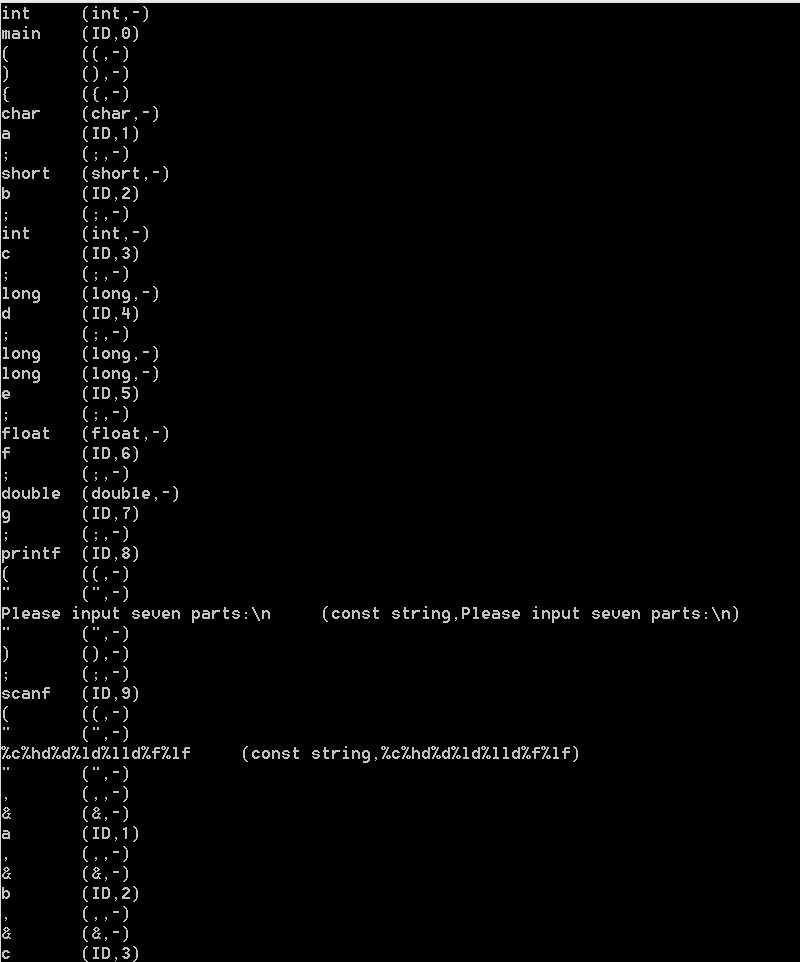
printf("The 'float' variable is %f,it takes %d bytes.\n",f,sizeof(f));

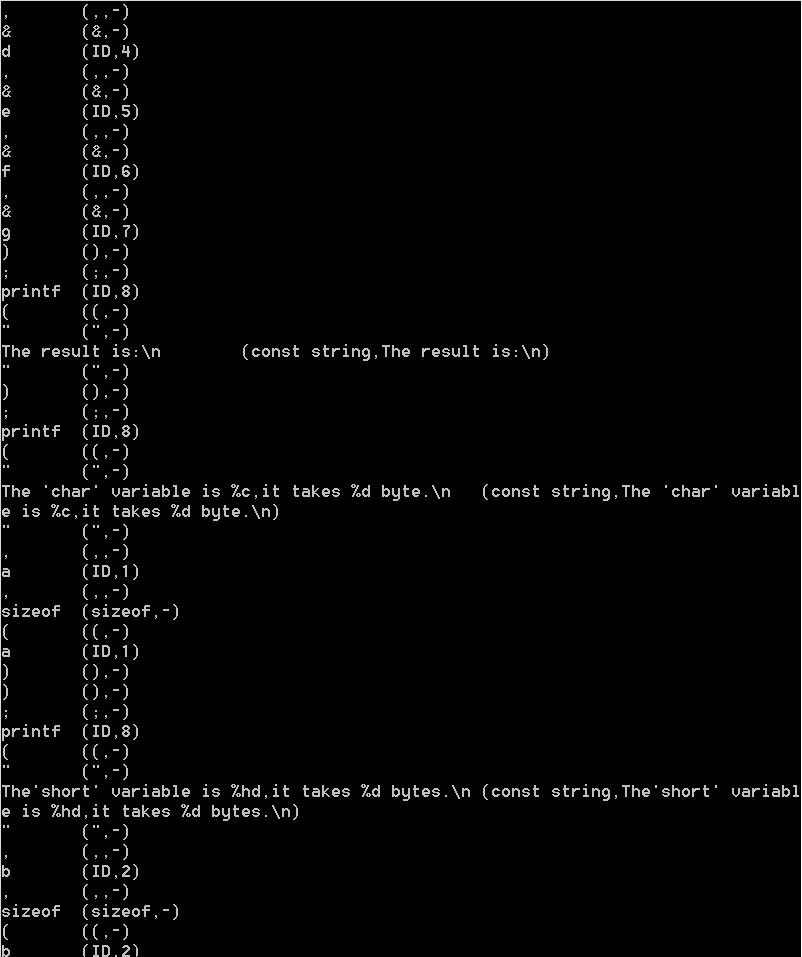
printf("The 'double variable' is %lf,it takes %d bytes.\n",g,sizeof(g));

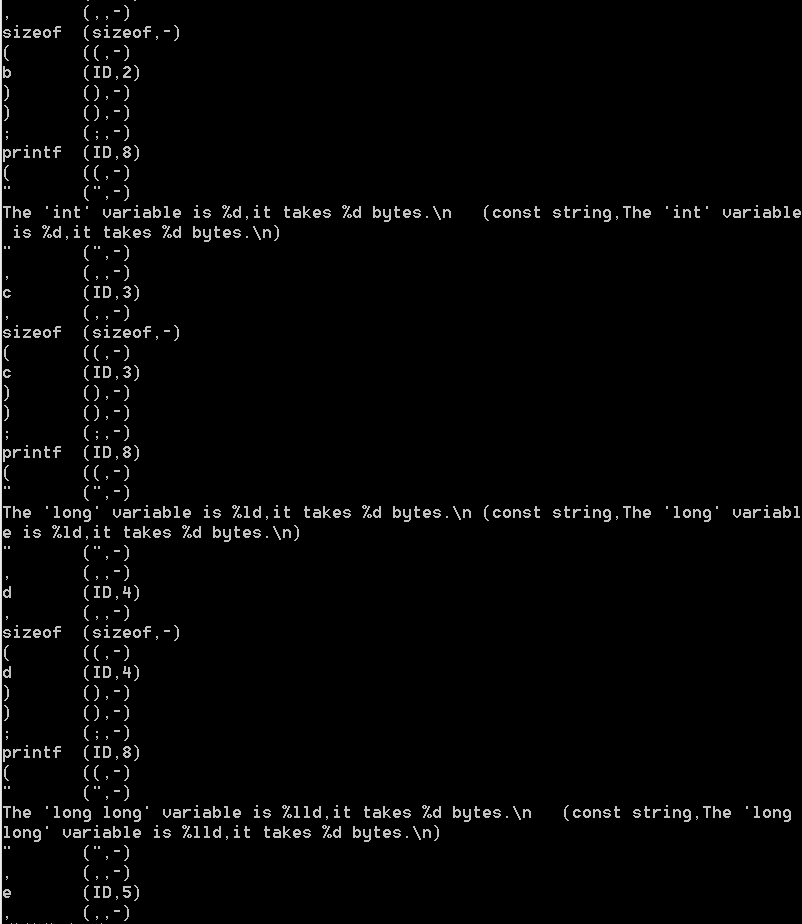
return 0;

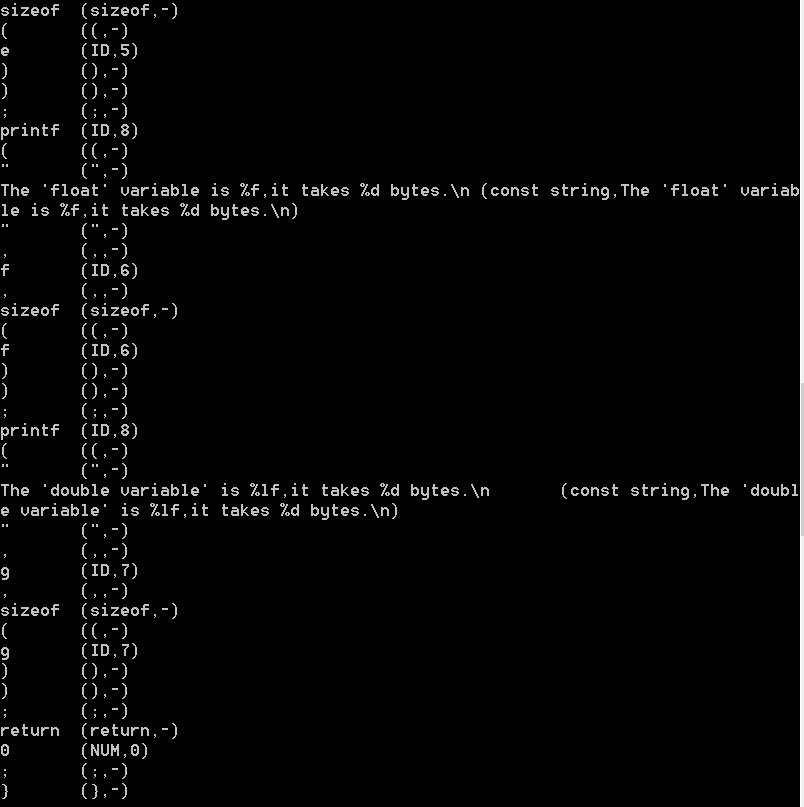
}

输出：











分析说明：正确进行词法分析和结果统计。

2、

输入：

#include<stdio.h>

int main()

{

double a,b,c,d,result;

printf("Please input four float numbers:");

scanf("%lf%lf%lf%lf",&a,&b,&c,&d);

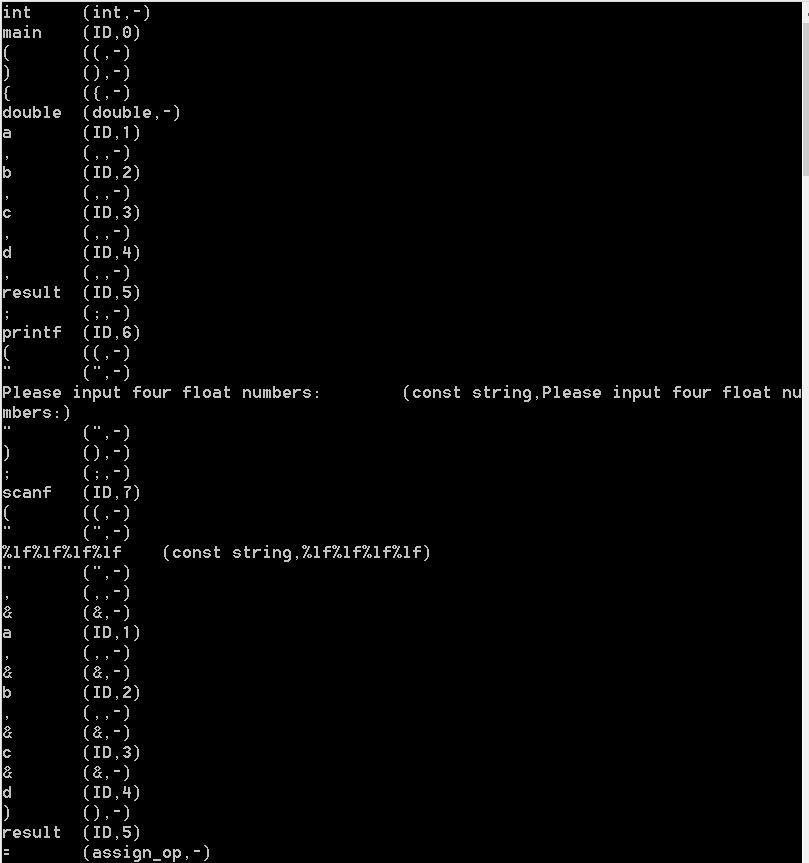
result=(a+b)\*(a-b)+c/d;

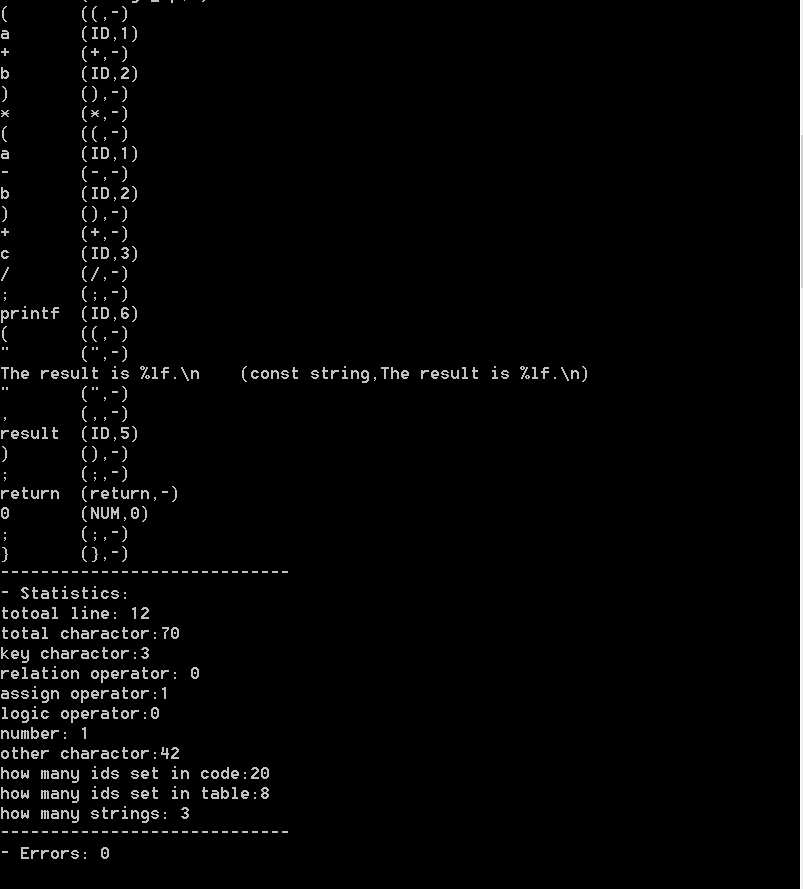
printf("The result is %lf.\n",result);

return 0;

}

输出：





分析说明：正确进行词法分析和结果统计。

3、

输入：

#include<stdio.h>

#include <stdlib.h>

int main()

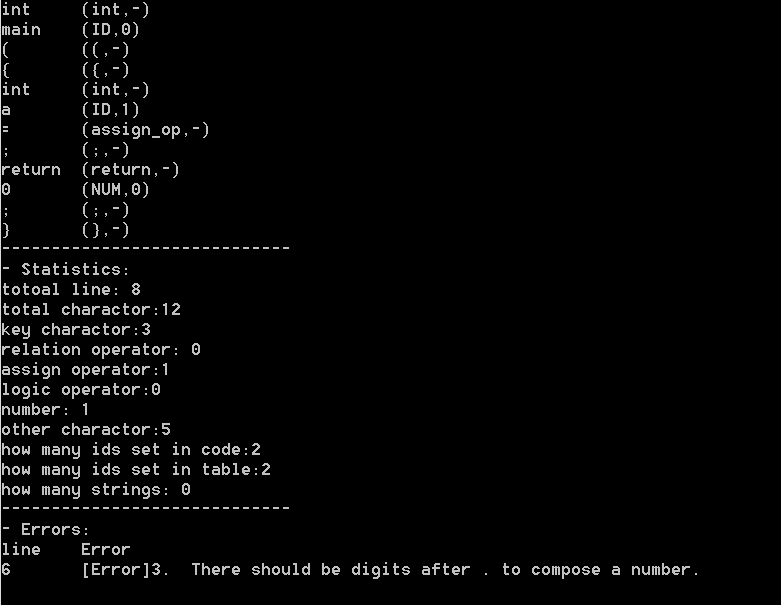
{

int a = 3.; /\*哈哈哈\*/

return 0; //哎呀呀

}

输出：



分析说明：正确进行词法分析和结果统计。