**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，要求如下。

（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）、可以识别并读取源程序中的注释。

（3）、可以统计源程序汇总的语句行数、单词个数和字符个数，其中标点和空格不计算为单词，并输出统计结果

（4）、检查源程序中存在的错误，并可以报告错误所在的行列位置。

（5）、发现源程序中存在的错误后，进行适当的恢复，使词法分析可以继续进行，通过一次词法分析处理，可以检查并报告源程序中存在的所有错误。

实验要求：

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：通过编写LEX源程序，利用LEX软件工具自动生成词法分析程序。

算法思路：

首先通过遍历，统计行号和字符数，并记录所有的注释，同时每一行作为一句添加’\0’符号。

其次，再次读取，利用一个字符数组作为buffer保存一行的数据，在对其中的数据进行处理，完成之后再读下一行，跳过注释。

对于整行数据的处理，依靠空格将其分成单个单词再具体处理。

对于查错问题，我只能根据一些规则判定有错并记录。

源代码：

#include<iostream>

#include<fstream>

#include<vector>

#include<string>

#include<map>

#include <cctype>

using namespace std;

const int MAX\_INPUT\_BUFFER = 512;//默认输入缓存区

ifstream file;//打开文件

//vector<string>keyword;//预先记录关键字

map<string, int> keyword = {

{ "auto",0 },

{ "double",0 },

{ "int",0 },

{ "struct",0 },

{ "break",0 },

{ "else",0 },

{ "long",0 },

{ "switch",0 },

{ "case",0 },

{ "enum",0 },

{ "register",0 },

{ "typedef",0 },

{ "char",0 },

{ "extern",0 },

{ "return",0 },

{ "union",0 },

{ "const",0 },

{ "float",0 },

{ "short",0 },

{ "unsigned",0 },

{ "continue",0 },

{ "for",0 },

{ "signed",0 },

{ "void",0 },

{ "default",0 },

{ "goto",0 },

{ "sizeof",0 },

{ "volatile",0 },

{ "do",0 },

{ "if",0 },

{ "while",0 },

{ "static",0 }

};

vector<string>comment;//存储注释

map<string, int> id;//标识符

vector<string>pre;//预处理文件

bool in\_comment = false;//标识 此行是否在注释中

int sum\_char = 0;//字符数

int sum\_word = 0;//单词数

int sum\_line = 0;//行数

int current\_line = 0;//当前行数

char buffer[MAX\_INPUT\_BUFFER];//缓存区

int big\_bracket = 0;//大括号

int small\_bracket = 0;//小括号

void sum(void)//计算字符数 行数 存注释

{

file.open("data.c");

int state = 0;//状态转换 0:正常 1:输入一个/ 2:输入/\*

char temp;//暂存

string temp\_comment;//暂存注释

file.seekg(0);//文件指针回到头

while (!file.eof())//遍历全文

{

switch (state)

{

case 0:

file.get(temp);

sum\_char++;//增加字符

if ('\n' == temp)

sum\_line++;//增加行

if ('/' == temp)

state = 1;

break;

case 1://前一个字符是 /

file.get(temp);

sum\_char++;//增加字符

if ('/' == temp)//单行注释

{

temp\_comment.clear();//清空注释缓存

while ('\n' != temp)//输完整行

{

file.get(temp);

sum\_char++;//增加字符

temp\_comment.append(1, temp);//添加注释

}

sum\_line++;//行增加

comment.push\_back(temp\_comment);//添加到注释记录表

state = 0;//状态回归

}

else

if ('\*' == temp)//多行注释

{

temp\_comment.clear();

state = 2;

}

else//不是注释

{

if ('\n' == temp)

sum\_line++;//增加行

state = 0; //状态回归

}

break;

case 2:

file.get(temp);

sum\_char++;//增加字符

if ('\*' == temp)

{

file.get(temp);//再取一个

sum\_char++;//增加字符

if ('/' == temp)//结束多行注释

{

comment.push\_back(temp\_comment);//存入注释表

state = 0;

}

else//还在注释中

{

temp\_comment.append(1, '\*');

temp\_comment.append(1, temp);

if ('\n' == temp)

sum\_line++;

}

}

else

{

if (file.eof())//当注释到尾时

{

if (!temp\_comment.empty())//不为空时存入最后一个注释

{

comment.push\_back(temp\_comment);

return;

}

}

if ('\n' == temp)

sum\_line++;

temp\_comment.append(1, temp);

}

break;

default:

break;

}

}

}

void word\_analyse(void)

{

string temp\_word;//暂存单词

char \*ptr = buffer;//处理指针

while (in\_comment)//多行注释问题

{

if ('\0' == \*ptr)

return;

if (('\*' == \*ptr) && ('/' == \*(ptr + 1)))

{

ptr += 2;

in\_comment = false;

}

ptr++;

}

while ('\0' != \*ptr)

{

if (' ' == \*ptr)//消空格

ptr++;

else if (isalpha(\*ptr) || '\_' == \*ptr)//关键字和标识符

{

temp\_word.clear();

while (isalnum(\*ptr) || '\_' == \*ptr)

{

temp\_word.append(1, \*ptr);

ptr++;

}//空格或其他字符时弹出

sum\_word++;

map<string, int>::iterator ltr;

ltr = keyword.find(temp\_word);

if (ltr != keyword.end()) {

ltr->second++;

cout << endl << "第" << sum\_word << "个单词：" << " 保留字：" << temp\_word << endl;

}

else {

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

cout << endl << "第" << sum\_word << "个单词：" << " 标识符：" << temp\_word << endl;

}

}

else if (isdigit(\*ptr))//数字

{

temp\_word.clear();

while (isdigit(\*ptr))

{

temp\_word.append(1, \*ptr);

ptr++;

}

if (isalpha(\*ptr) || '\_' == \*ptr)

{

while (' ' == \*ptr && '\0' == \*ptr)

{

temp\_word.append(1, \*ptr);

ptr++;

}

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 非法字符：" << temp\_word << endl;

cout << "error:在第" << current\_line << "行,单词" << temp\_word << "命名非法！" << endl;

}

else

{

if ('.' == \*ptr || 'E' == \*ptr || 'e' == \*ptr)//小数和指数形式

{

temp\_word.append(1, \*ptr);

ptr++;

while (isdigit(\*ptr))

{

temp\_word.append(1, \*ptr);

ptr++;

}

}

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 无符号数：" << temp\_word << endl;

}

}

else if ('#' == \*ptr)//预处理文件

{

while ('\0' != \*ptr)

{

temp\_word.append(1, \*ptr);

ptr++;

}

pre.push\_back(temp\_word);

}

else if ('"' == \*ptr)//字符串

{

temp\_word.clear();

ptr++;

while ('"' != \*ptr)

{

temp\_word.append(1, \*ptr);

ptr++;

}

ptr++;

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 字符串：" << temp\_word << endl;

}

else if ('+' == \*ptr)

{

temp\_word.clear();

if ('+' == \*(ptr + 1))

{

temp\_word = "++";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 加等于号：" << temp\_word << endl;

ptr += 2;

}

else if ('=' == \*(ptr + 1))

{

temp\_word = "+=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 自加号：" << temp\_word << endl;

ptr += 2;

}

else if (isdigit(\*(ptr + 1)))//有符号数

{

temp\_word = "+";

for (int j = 1; isdigit(\*(ptr + j)); j++)

temp\_word.append(1, \*(ptr + j));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 有符号数：" << temp\_word << endl;

ptr += 2;//推进

}

else

{

temp\_word = "+";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 加号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if ('-' == \*ptr)

{

temp\_word.clear();

if ('-' == \*(ptr + 1))

{

temp\_word = "--";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 自减号：" << temp\_word << endl;

ptr += 2;

}

else if ('=' == \*(ptr + 1))

{

temp\_word = "-=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 减等于号：" << temp\_word << endl;

ptr += 2;

}

else if ('>' == \*(ptr + 1))

{

temp\_word = "->";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 指针号：" << temp\_word << endl;

ptr += 2;

}

else if (isdigit(\*(ptr + 1)))//有符号数

{

temp\_word = "-";

for (int j = 1; isdigit(\*(ptr + j)); j++)

temp\_word.append(1, \*(ptr + j));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 有符号数：" << temp\_word << endl;

ptr += 2;//推进

}

else

{

temp\_word = "-";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 减号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if ('\*' == \*ptr)

{

temp\_word.clear();

if ('=' == \*(ptr + 1))

{

temp\_word = "\*=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 乘等于号：" << temp\_word << endl;

ptr += 2;

}

else

{

temp\_word = "\*";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 乘号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if ('/' == \*ptr)

{

temp\_word.clear();

if ('=' == \*(ptr + 1))

{

temp\_word = "/=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 除等于号：" << temp\_word << endl;

ptr += 2;

}

else if ('/' == \*(ptr + 1))

return;//行注释直接跳出

else if ('\*' == \*(ptr + 1))//多行注释

{

in\_comment = true;

ptr += 2;

while ('\0' != \*ptr)//此行结束注释是否也结束

{

if (('\*' == \*ptr) && ('/' == \*(ptr + 1)))

{

in\_comment = false;

break;

}

ptr++;

}

}

else

{

temp\_word = "/";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 除号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if ('=' == \*ptr)

{

temp\_word.clear();

if ('=' == \*(ptr + 1))

{

temp\_word = "==";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 相等比较号：" << temp\_word << endl;

ptr += 2;

}

else

{

temp\_word = "=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 赋值号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if ('>' == \*ptr)

{

temp\_word.clear();

if ('=' == \*(ptr + 1))

{

temp\_word = ">=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 大于等于号：" << temp\_word << endl;

ptr += 2;

}

else

{

temp\_word = ">";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 大于号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if ('<' == \*ptr)

{

temp\_word.clear();

if ('=' == \*(ptr + 1))

{

temp\_word = "<=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 小于等于号：" << temp\_word << endl;

ptr += 2;

}

else

{

temp\_word = "<";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 小于号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if ('!' == \*ptr)

{

temp\_word.clear();

if ('=' == \*(ptr + 1))

{

temp\_word = "!=";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 不等于号：" << temp\_word << endl;

ptr += 2;

}

else

{

temp\_word = "!";

map<string, int>::iterator ltr;

ltr = id.find(temp\_word);

if (ltr != id.end())

ltr->second++;

else

id.insert(pair<string, int>(temp\_word, 1));

sum\_word++;

cout << endl << "第" << sum\_word << "个单词：" << " 取反号：" << temp\_word << endl;

ptr += 2;//推进

}

}

else if (':' == \*ptr || '(' == \*ptr || ')' == \*ptr || ';' == \*ptr || '{' == \*ptr || '}' == \*ptr || ',' == \*ptr

|| '[' == \*ptr || ']' == \*ptr || '\0' == \*ptr || '\n' == \*ptr)//标点不算单词

{

if ('(' == \*ptr)

small\_bracket++;//查错

if (')' == \*ptr)

small\_bracket--;//查错

if ('{' == \*ptr)

big\_bracket++;//查错

if ('}' == \*ptr)

big\_bracket--;//查错

ptr++;//推进

}

else

{

cout << endl << "错误:在第" << current\_line << "行有错误：" << \*ptr << endl;

ptr++;

}

}

}

void analyse(void)

{

char \*p = buffer;

file.seekg(0);

while (!file.eof())

{

file.get(\*p);

if ('\n' == \*p)//一句

{

\*(p + 1) = '\0';

\*(p + 2) = '\0';

current\_line++;

word\_analyse();

if (small\_bracket != 0)

{

cout << endl << "错误:在第" << current\_line << "行小括号不匹配" << endl;

small\_bracket = 0;

}

p = buffer;//重新填充

}

else

p++;

}

if (big\_bracket != 0)

cout << endl << "错误:大括号不匹配！" << endl;

}

void show(void)

{

cout << endl;

map<string, int>::iterator ltr;

for (ltr = keyword.begin(); ltr != keyword.end(); ltr++)

if (ltr->second != 0)

cout << ltr->first << "\t" << ltr->second << endl;

for (ltr = id.begin(); ltr != id.end(); ltr++)

cout << ltr->first << "\t" << ltr->second << endl;

cout << endl << "单词总数:" << sum\_word << endl;

cout << endl << "字符总数:" << sum\_char << endl;

cout << endl << "总行数is:" << sum\_line << endl;

cout << "头文件有:" << endl;

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

{

cout << pre.at(i);

}

cout << "注释有:" << endl;

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

{

cout << comment.at(i);

}

cout << endl;

}

int main(void)

{

file.open("data.c");

if (!file)

cout << endl << "找不到文件！" << endl;

else

{

//开始处理

analyse();

file.close();

sum();//统计行数 字节数 注释

show();//显示统计结果

}

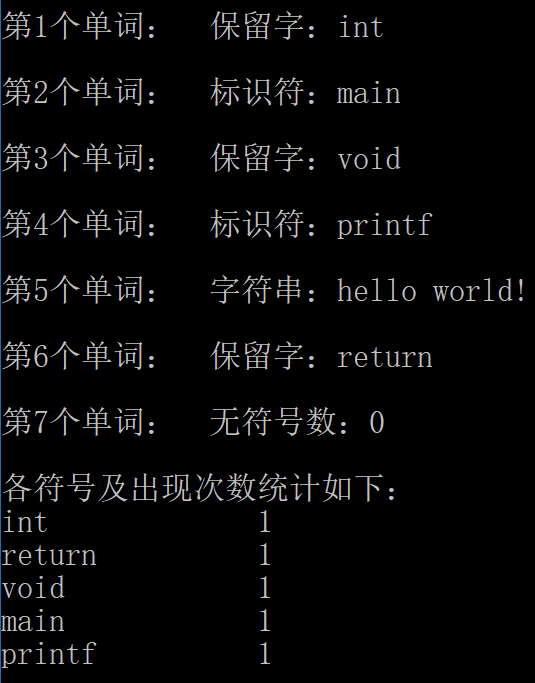
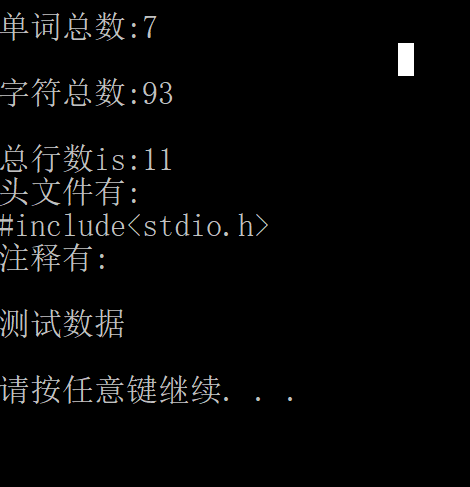
system("pause");

return 0;

}

结果分析：

**正确程序测试**

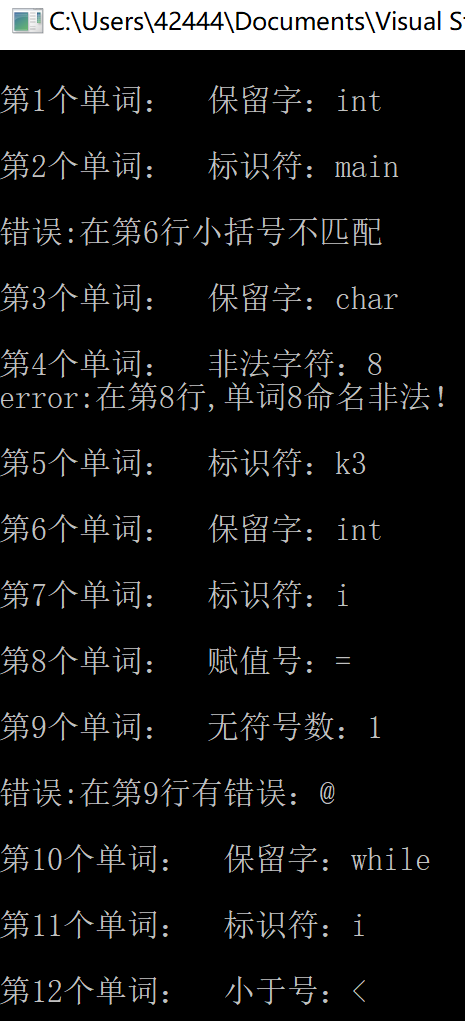
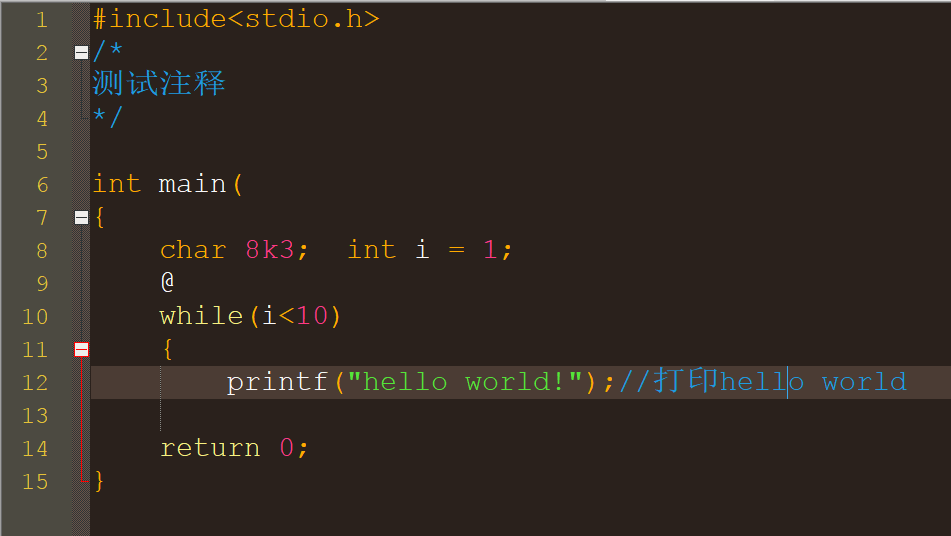
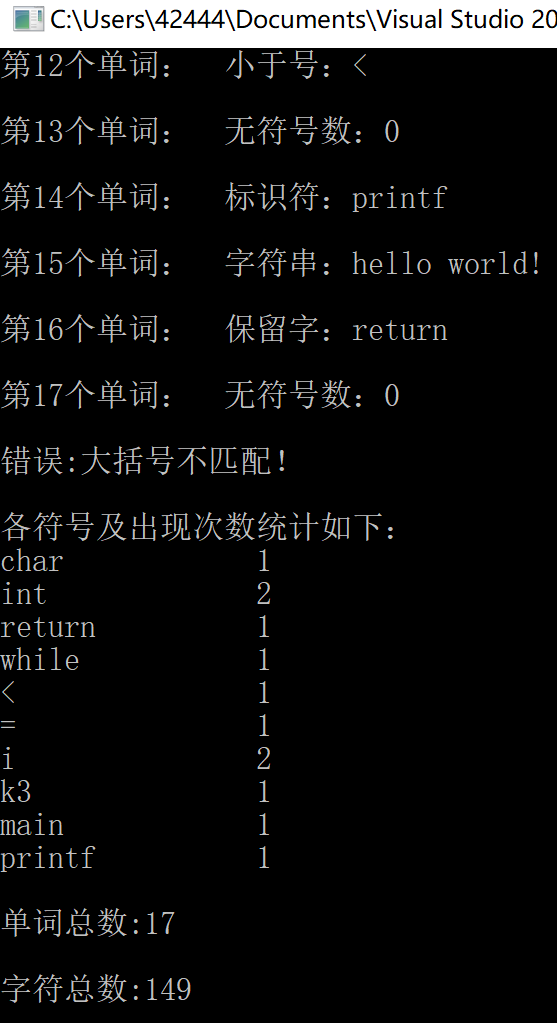
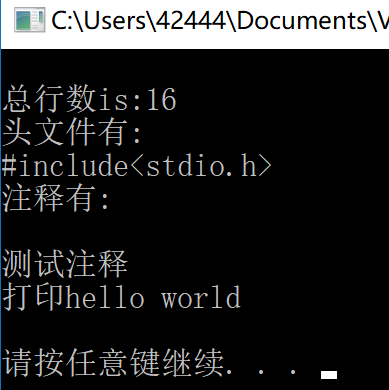
  

字符数包括空格

**复杂程序测试**

data.c，由于太多无法截图，可以自行尝试。

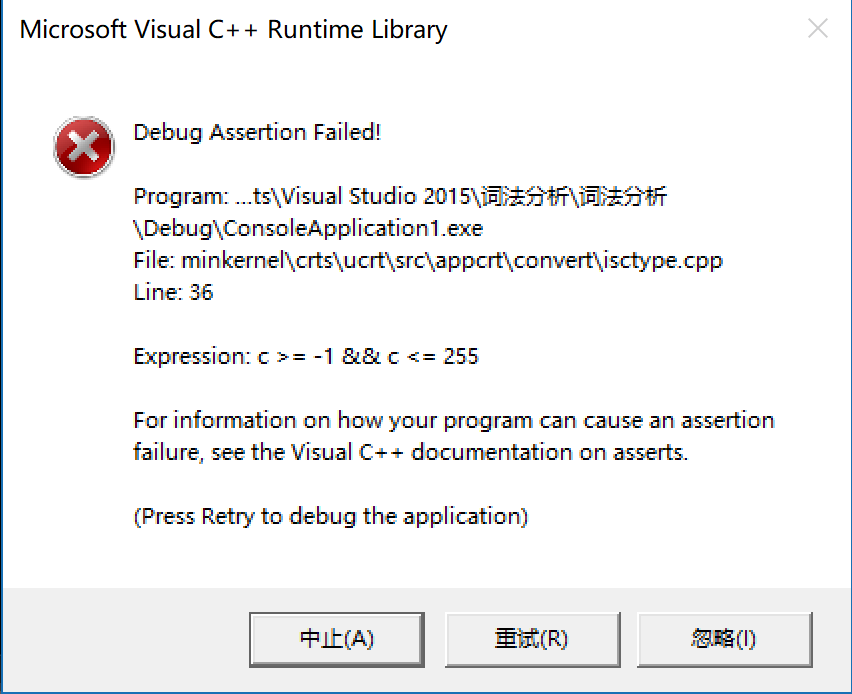
**有错误程序测试**



字符数包括空格，错误处理方面如命令行第二个红圈所示，将可以构成标识符的作为一个变量

实验反思：

1. 本次实验可以说我很清楚的了解了词法分析的流程，对于这种我们无法直接看到的过程有了概念。
2. 代码中的问题有不少，我觉得最大的一点就是这次强迫自己用了c++的独特知识，开始本来想用vector就够了，但要求中有说需要输出各个字符的出现次数，无奈搜索很久没找到vector计数（可能是傻了），但是搜到了另一种c++容器——map，而这种key-value的组合深得我心，而且操作简单，同时据了解map的搜索效率也很高，所以可以说这次的一大收获就是学会了map的相关操作。
3. 代码中还有不足就是格式上没有采用自动机，尽管我自己花了不少时间画了一个自动机的状态转移图，但真正写的时候。。。脑子就抽风了，写着写着就写成了if-else的分支语句。
4. 程序中其实讲实话情况没有考虑的很完美，至少想到去看C99的标准但是最后因为时间快到所以就没改，会在课下改完的（嗯）。
5. 代码的注释可以看到前面挺详尽但后面。。。。实在不想写了，打代码一有状态打好多，然后就不想打注释了。
6. 代码的思路在第一页提到了，感觉和一些同学的不同，因为当时由于选用vector和map的问题请教同学的时候= =居然被说代码逻辑看不懂。。。我觉得我这个很傻瓜的逻辑啊，所以可能以后要想的更详尽更仔细一点。
7.  这个问题开始没有出现，但写这个报告的时候就出现了，查阅博客这个问题是isalpha参数中是无符号数应该强制转换为成unsigned，但是之后没有用……后来想起了两个小时之前自己作死的把代码传到github的时候吧编码格式改了（因为github上乱码），当改回来的时候就没有问题了……很迷。