



北京交通大学
BEIJING JIAOTONG UNIVERSITY

《编译原理》专题实验一

学 号： 17281033

姓 名： 贡乐天

专 业： 计算机科学与技术

学 院： 计算机与信息技术学院

指导老师： 徐金安

提交日期： 2020 年 4 月 3 日

目录

专题 1 词法分析程序设计实验报告	1
一、 实验题目与要求	1
1.1 实验内容.....	1
1.2 实验要求.....	1
1.3 实验环境（开发/运行/测试）	2
二、 程序功能描述	2
三、 数据结构及函数描述	2
四、 程序结构描述	8
4.1 单词符号类别编码表.....	8
4.2 主要函数说明.....	10
4.3 函数调用关系.....	10
4.4 DFA 状态转化图.....	11
4.5 执行框图.....	12
五、 程序测试	13
六、 实验总结	15
七、 程序清单	16

专题 1 词法分析程序设计实验报告

一、实验题目与要求

1.1 实验内容

以下为正则文法所描述的 C 语言子集单词符号的示例，请补充单词符号：
++，--，>>，<<，+=，-=，*=，/=，&&（逻辑与），||（逻辑或），！（逻辑非）等等，给出补充后描述 C 语言子集单词符号的正则文法，设计并实现其词法分析程序。

<标识符>→字母 | <标识符>字母 | <标识符>数字

<无符号整数>→数字 | <无符号整数>数字

<单字符分界符> →+ | - | * | ; | , | (|) | { | }

<双字符分界符>→<大于>= | <小于>= | <小于>> | <感叹号>= | <等于>= | <斜竖>*

<小于>→< <等于>→=

<大于>→> <斜竖> →/

<感叹号>→!

该语言的保留字：void、int、float、double、if、else、for、do、while 等等（也可补充）。

设计说明：

（1）可将该语言设计成大小写不敏感，也可设计成大小写敏感，用户定义的标识符最长不超过 32 个字符；

（2）字母为 a-z A-Z，数字为 0-9；

（3）可以对上述文法进行扩充和改造；

（4）“/*……*/”和“//”（一行内）为程序的注释部分。

1.2 实验要求

（1）给出各单词符号的类别编码；

（2）词法分析程序应能发现输入串中的错误；

（3）词法分析作为单独一遍编写，结果为二元式序列组成的中间文件；

（4）设计两个测试用例（尽可能完备），并给出测试结果。

1.3 实验环境（开发/运行/测试）

表 1-1 实验开发/运行/测试环境

	名称	版本号
OS	Windows	10
编译器	VS	2019

二、程序功能描述

- 给出了各单词符号的类别编码；
- 词法分析程序能够对给出的文件中的输入串做出正确的词法；
- 词法分析程序能发现文件输入串中的错误；
- 词法分析结果为二元式序列组成的中间文件；
- 词法分析程序能兼容注释并能发现文件注释未关闭的错误；
- 正确识别换行符，并在输出文件进行相应换行。

三、数据结构及函数描述

为了表征 C 语言中可能出现的单词、分隔符、操作符等等，我定义了如下的数据结构，用以在程序内部表征各种情况的内部编码，为了方便错误处理与打印相关信息，我定义了对每个较为单独的"单词"、"数字"及"操作符"的分类。下面详细介绍我设计的数据结构及内容。

- 属性类，包括分界符标签、单词符号种别码以及单词符号字符串

```
typedef struct restype {  
    int tag;          /*分界符标签*/  
    string num;       /*种别码*/  
    string word;      /*单词符号*/  
};
```

- isWord () // 判断单词符号的种别码
 - Function: // isWord
 - Description: // 判断单词符号的种别码
 - Calls: // 无
 - Input: // 待判断的单词符号
 - Output: // 无输出参数
 - Return: // 返回 restype 类型变量
 - Others: // tag 为 0 表示不是分界符，为 1 表示是分界符

```
restype isWord(string word) {
    restype type;
    type.tag = 0; //确定不是分界符
    //全部转化为小写
    for (int i = 0; i < word.length(); ++i) {
        if (word[i] >= 'A' && word[i] <= 'Z') {
            word[i] = word[i] + 32;
        }
    }
    //保留字
    if (word == "void") { type.num = "1"; type.word = word; }
    else if (word == "int") { type.num = "2"; type.word = word; }
    .....
    .....
    .....
    else if (word == "break") { type.num = "11"; type.word = word; }
    else {
        int state = 1;
        for (int j = 0; j < word.length() && state == 1; j++) {
            if (word[j] == '0' || word[j] == '1' || word[j] == '2'
|| word[j] == '3' || word[j] == '4' || word[j] == '5' || word[j] ==
'6' || word[j] == '7' || word[j] == '8' || word[j] == '9');
                else { state = 0; }
            }
        //数字
        if (state == 1) {
            type.num = "13"; type.word = word;
        }
        //未标识
        else {
            type.num = "12"; type.word = word;
        }
    }
    return type;
}
```

- isDelimiter () // 判断单个字符为单分界符
 - Function: // isDelimiter
 - Description: // 判断单个字符为单分界符
 - Calls: // 无
 - Input: // 待判断字符
 - Output: // 无输出参数
 - Return: // 返回 restype 类型变量
 - Others: // tag 为 0 表示不是分界符，为 1 表示是分界符

```
restype isDelimiter(char word) {
    restype type;
    type.tag = 0; // 初始化为不分界符
    type.num = "0";
    char c1 = word;
    type.word = c1;

    if (c1 == '+') { type.tag = 1; type.num = "14"; type.word = c1; }
    else if (c1 == '-') { type.tag = 1; type.num = "15"; type.word
= c1; }
    else if (c1 == '*') { type.tag = 1; type.num = "16"; type.word
= c1; }

    .....
    .....
    .....

    else if (c1 == '|') { type.tag = 1; type.num = "34"; type.word
= c1; }
    else if (c1 == '\n') { type.tag = 1; type.num = "35"; type.word
= c1; }

    return type;
}
```

- analyse () // 词法分析主体函数，对 readTxt () 传参的全部字符串进行分析，并将最终结果以 string 形式传递给 writeResult () 函数。分析主要思想是先获得单词符号，再进行判断。

- Function: // analyse

- Description: // 词法分析主体函数
- Calls: // isDelimiter; isWord;
- Input: // 从文件中读取的原字符串
- Output: // 无输出参数
- Return: // 返回得到二元式序列组成的结果字符串
- Others: // 完成词法分析，与注释未关闭导致的错误的检测

```

string analyse(string my_str) {
    string result_str = ""; //记录分析结果
    string word = "";
    //获取词
    my_str = my_str + ' ';
    int length = my_str.length();
    for (int i = 0; i < length; i++) {
        if (my_str[i] != ' ' && my_str[i] != '\r\t' && my_str[i] !=
            '\0' && isDelimiter(my_str[i]).tag == 0 && (i + 1) != length)
        {
            word = word + my_str[i];
        }
        else { //开始分析
            if (word != "") {
                restype judge1 = isWord(word);
                result_str = result_str + "(" + judge1.num + "," +
                    judge1.word + ")";
                word = ""; //对 word 进行清空,以便于记录下一个 word
            }
            //对当前分界符进行分析
            restype judge2 = isDelimiter(my_str[i]);
            if (judge2.tag == 1) {
                if (judge2.num == "21" && my_str[i + 1] == '=') {
                    result_str = result_str + "(41, >=)";
                    i++; //跳过双分界符的第二个字符
                }
                else if (judge2.num == "23" && my_str[i + 1] == '=')
            {
                result_str = result_str + "(42, <=)";
                i++; //跳过双分界符的第二个字符
            }

            .....
            .....
            .....

```

```
else if (judge2.num == "30" && my_str[i + 1] == '=')
{
    result_str = result_str + "(55, !=)";
    i++; //跳过双分界符的第二个字符
}
//*****对换行符进行处理*****
else if (judge2.num == "35" ) {
    result_str = result_str + "\n";
}
//*****对注释进行处理*****
else if (judge2.num == "28" && my_str[i + 1] == '*')
{
    result_str = result_str + "(注释开始, /*)";
    i++; //跳过双分界符的第二个字符
    int state = 0;
    int j = i + 1;
    for (; j < length && state == 0; j++) {
        if (my_str[j] == '*' && my_str[j + 1] == '/')
        {
            state = 1;
        }
    }
    if (state == 1) { //注释有结尾
        result_str = result_str + "(注释关闭, */)";
        i = j;
    }
    else {
        result_str = result_str + "(出现错误, 注释未关闭!)";
        i = length;
    }
} //开始分析
else {
    result_str = result_str + "(" + judge2.num + ", " + judge2.word + " )";
}
} //获取词
}
}
return result_str;
}
```


- writeResult () // 将全部的分析结果写入文件
 - Function: // writeResult
 - Description: // 将分析结果写入文件
 - Calls: // 无
 - Input: // 结果字符串
 - Output: // 结果字符串写入结果文件

```
void writeResult(string result_str) {
    ofstream out("output1.txt");
    if (out.is_open())
    {
        out << result_str << endl;
        out.close();
    }
}
```

- readTxt () // 读取文件中的全部字符串，并将字符串传给 analyse
() 函数进行分析

- Function: // readTxt
- Description: // 读取文件中的字符串

```
string readTxt(string file)
{
    ifstream infile;
    infile.open(file.data()); //将文件流对象与文件连接起来
    assert(infile.is_open()); //若失败,则输出错误消息,并终止程
序运行
    if (!infile) {
        cerr << "读取文件出错了!" << endl;
        exit(1);
    }
    char ch;
    string my_str = "";
    infile >> noskipws;
    while (infile.get(ch))//从文件读取字符进来
    {
        my_str += ch;
    }
    infile.close(); //关闭文件输入流
    return my_str;
}
```

- `main()`//主函数

```
string my_str = readTxt("input2.txt");  
string result_str = analyse(my_str);  
writeResult(result_str);  
  
return 0;
```

四、程序结构描述

4.1 单词符号类别编码表

单词符号	种别码
void	1
int	2
float	3
double	4
if	5
else	6
for	7
do	8
while	9
return	10
break	11
未标识	12

单词符号	种别码
无符号整数	13
+	14
-	15
*	16
;	17
(18
)	19
=	20
>	21
<	23
:	26
/	28
!	30
{	31
}	32
&	33

单词符号	种别码
	34
\n	35
>=	41
<=	42
<>	43
:=	44
+=	45
++	46
-=	47
--	48
*=	49
/=	50
&&	51
	52
>>	53
<<	54
!=	55

4.2 主要函数说明

isDelimiter () // 判断单个字符为单分界符

isWord () // 判断单词符号的种别码

analyse () // 词法分析主体函数，对 readTxt () 传参的全部字符串进行分析，并将最终结果以 string 形式传递给 writeResult () 函数。分析主要思想是先获得单词符号，再进行判断。

writeResult () // 将全部的分析结果写入文件

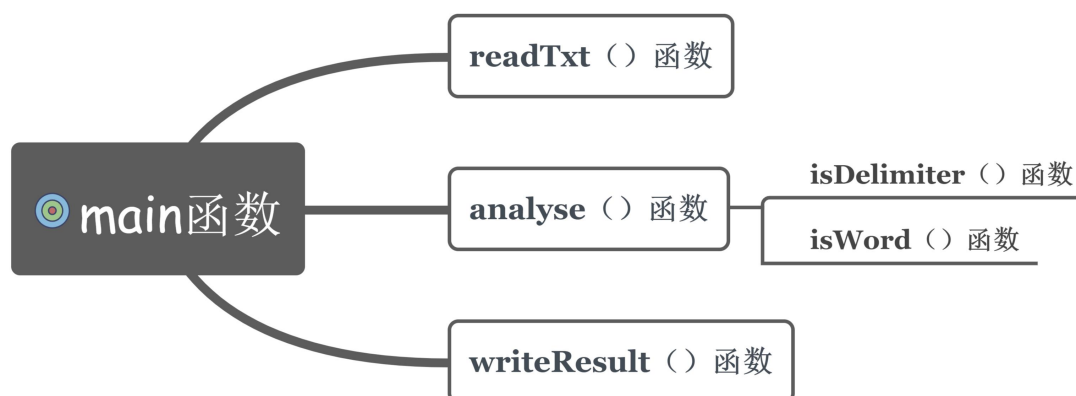
readTxt () // 读取文件中的全部字符串，并将字符串传给 analyse () 函数进行分析

main()//主函数

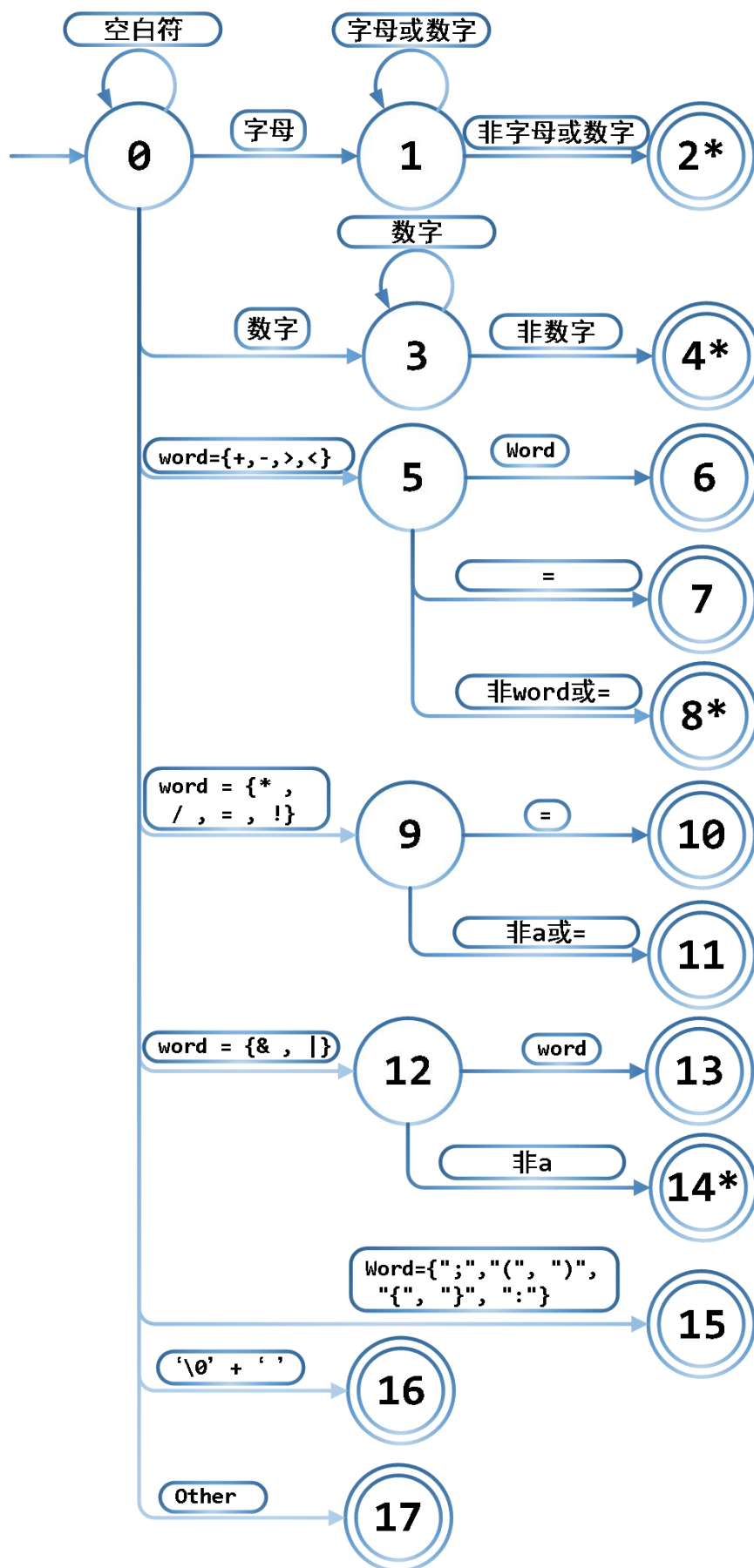
4.3 函数调用关系

【1】main () 函数调用：readTxt () 函数、analyse () 函数、writeResult () 函数；

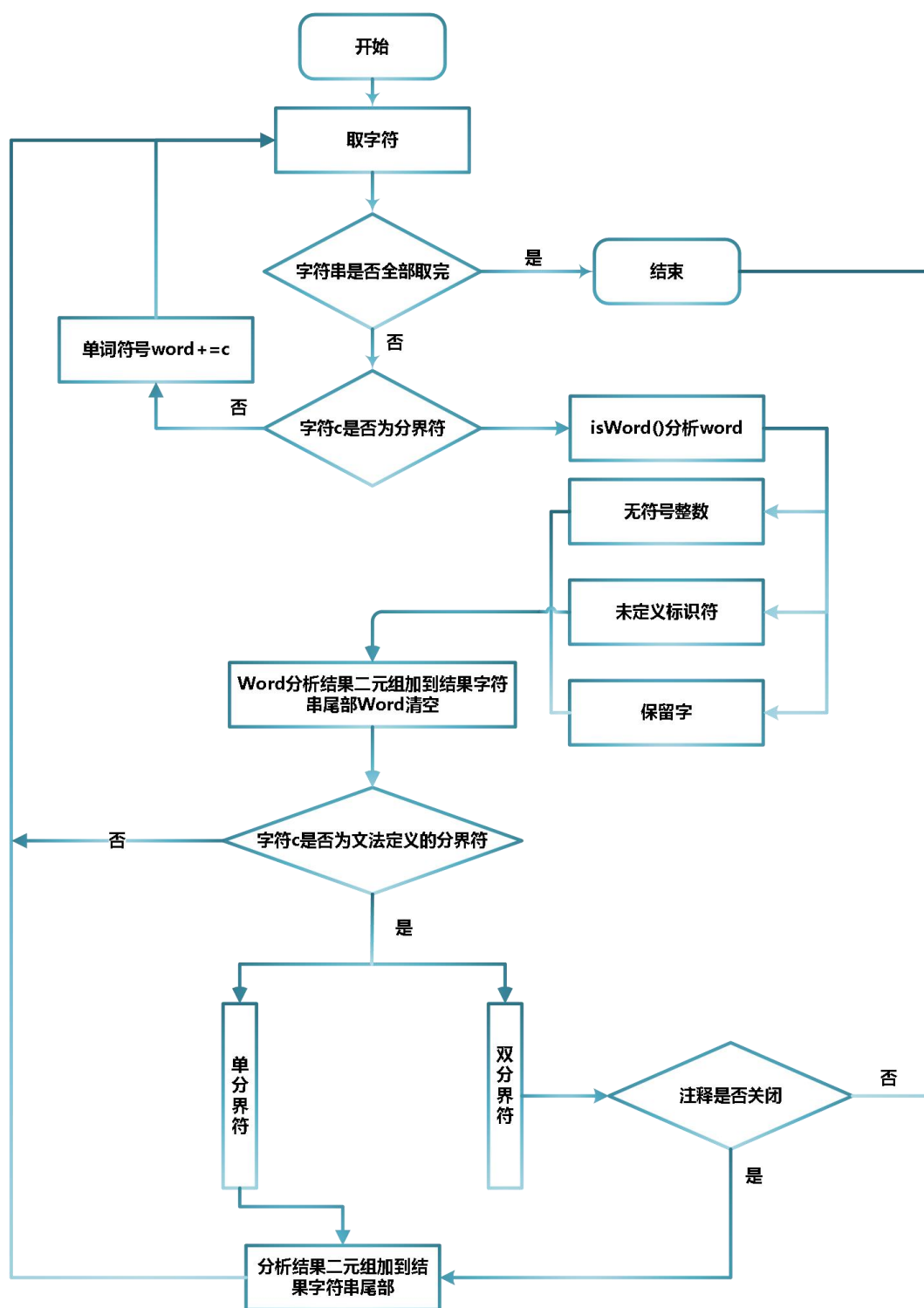
【2】analyse () 函数调用：isDelimiter () 函数、isWord () 函数



4.4 DFA 状态转化图



4.5 执行框图



五、程序测试

【测试代码 1】

```

input1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
void main() { /*GongLetian-17281033*/
    int sum=0;
    for(int i = 0;i<=10;i++) {
        if (i!=3||i!=6) {
            sum+=i<<2;
            sum+=i>>2;
            sum +=2;
            sum *=2;
            sum /=2;
            sum -=2;
        }
        if(sum==63&& i<5||sum==128)
            break;
    }

    return 0;
}

/*GongLetian-17281033*/

```

【测试结果 1】

```

output1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
(1,void)(12,main)(18, )(19, )(31, { }(注释开始, /* )(注释关闭, */ )
(2,int)(12,sum)(20, = )(13,0)(17, ; )
(7,for)(18, )(2,int)(12,i)(20, = )(13,0)(17, ; )(12,i)(42, <= )(13,10)(17, ; )(12,i)(46, ++ )(19, ) )(31, { )
(5,if)(18, )(12,i)(55, != )(13,3)(52, || )(12,i)(55, != )(13,6)(19, ) )(31, { )
(12,sum)(45, += )(12,i)(54, << )(13,2)(17, ; )
(12,sum)(45, += )(12,i)(53, >> )(13,2)(17, ; )
(12,sum)(45, += )(13,2)(17, ; )
(12,sum)(49, *= )(13,2)(17, ; )
(12,sum)(50, /= )(13,2)(17, ; )
(12,sum)(47, -= )(13,2)(17, ; )
(32, } )
(5,if)(18, )(12,sum)(20, = )(20, = )(13,63)(51, && )(12,i)(23, < )(13,5)(52, || )(12,sum)(20, = )(20, = )(13,128)(19, ) )
(11,break)(17, ; )
(32, } )

(10,return)(13,0)(17, ; )
(32, } )

(注释开始, /* )(注释关闭, */ )

```

【测试代码 2】

```

input2.txt - 记事本
文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)
typedef struct restype {
    int tag;           /*分界符标签*/
    double num;        /*种别码*/
    float word;        /*单词符号*/
};
while (infile.get(ch))
{
    my_str += ch;
}
infile.close();
return my_str;
int aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```

【测试结果 2】

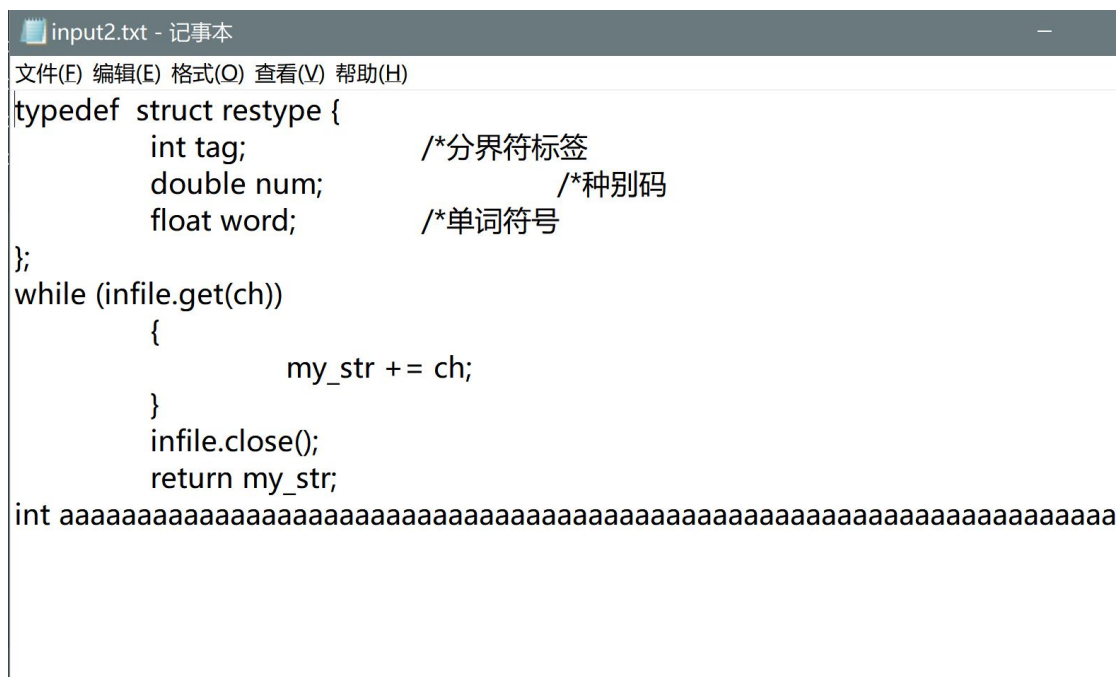
```

output2.txt - 记事本
文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)
(12,typedef)(12,struct)(12,restype)(31, { )
(12,      int)(12,tag)(17, ; )(12,                )(注释开始, /* )(注释关闭, */ )
(12,      double)(12,num)(17, ; )(12,                )(注释开始, /* )(注释关闭, */ )
(12,      float)(12,word)(17, ; )(12,                )(注释开始, /* )(注释关闭, */ )
(32, } )(17, ; )
(9,while)(18, ( )(12,infile.get)(18, ( )(12,ch)(19, ) )(19, ) )
(12,      )(31, { )
(12,      my_str)(45, += )(12,ch)(17, ; )
(12,      )(32, } )
(12,      infile.close)(18, ( )(19, ) )(17, ; )
(12,      return)(12,my_str)(17, ; )
(2,int)
用户定义的标识符最长不超过32个字符!

用户定义的标识符最长不超过32个字符!

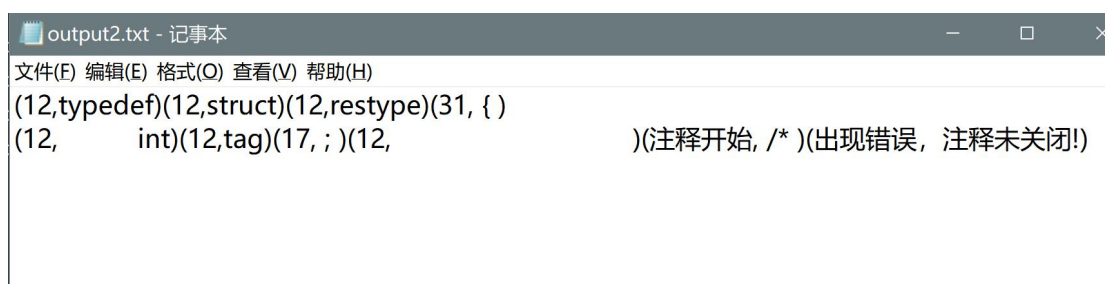
```


【测试代码 3】



```
typedef struct restype {
    int tag;          /*分界符标签
    double num;       /*种别码
    float word;       /*单词符号
};
while (infile.get(ch))
{
    my_str += ch;
}
infile.close();
return my_str;
int aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

【测试结果 3】



```
(12,typedef)(12,struct)(12,restype)(31, { )
(12, int)(12,tag)(17, ; )(12, )(注释开始, /* )(出现错误, 注释未关闭!)
```

六、实验总结

我在编写该程序是时的想法是先获得单词符号。如何获得每一个单词符号 word 呢？我以分界符为标准进行判断。分界符分为文法定义的分界符和非文法定义的分界符。当前字符 c 不是分界符时，则 $word += c$ ；若 c 为非文法定义的分界符，则说明此时一个单词符号已经结束，即为当前的 word。调用函数对 word 进行判断即可。若 c 为文法定义的分界符，继续对 c 进行判断，为单分界符或字符串(c 与 c 后面的一个字符)双分界符。此处对 “/” 进行查错分析。至此以一个分界符 c 为周期的分析结束，需要对 word 进行清空。然后开始下一个周期的分析，继续获取字符至下一个分界符。

此次实验通过自己手工实现一个简单的 C 语言的词法分析器，对编译原理课上学习的词法分析过程有了更深刻的认识。在编写程序过程中，通过对诸如读取、分析等步骤的深入思考，加深了对 C 语言本身词法的认识。在编写过程中，也学习到了很多设计技巧，如利用取模操作实现循环读取缓冲区等。设计程序框架时，根据操作系统课上学到的信号量知识，采用多线程编程，提高了程序运行的速度，同时保证进程有序的执行。

总的来说，此次研究性实验让我对词法分析器的工作原理与功能从感性认识到了具体实现。在编写过程中，遇到了很多非常具体的问题，通过一一解决它们，相信在以后实验中，以此为基础，能够更好的，更快的实现要求的功能。

通过本次实验我锻炼了自己的上机操作能力及编程能力，并对理论知识有了进一步的了解。本实验基本思路比较清晰，用较为简单的算法就能实现；解决实验中遇到的问题也花费了一部分时间，我增长了处理关于文件错误的能力；实验遇到的主要问题是在对分析周期的截取当中，在循环当中 i 与 j 的关系容易出错。

此程序实现了要求中的所有功能，并增加了对注释关闭的判断。但有的地方不免有些繁杂，还有一些潜藏的问题，需要进一步测试来使程序变得更加具有健壮性。

七、程序清单

```
/*  
Author: 贡乐天  
Date: 2020-04-03  
Description: 词法分析程序  
*/  
  
#include <iostream>  
#include <ctype.h>  
#include <string.h>  
#include <stdlib.h>  
#include <fstream>  
#include <cassert>  
#include <sstream> // 使用 istream 所需要的头文件  
using namespace std;  
  
/* 属性类，包括分界符标签、单词符号种别码以及单词符号字符串 */  
typedef struct restype {
```

```

    int tag;      /*分界符标签*/
    string num;    /*种别码*/
    string word; /*单词符号*/
};

/*****
Function:      // isDelimiter
Description:    // 判断单个字符为单分界符
Calls:         // 无
Input:         // 待判断字符
Output:        // 无输出参数
Return:        // 返回 restype 类型变量
Others:        // tag 为 0 表示不是分界符，为 1 表示是分界符
*****/
restype isDelimiter(char word) {
    restype type;
    type.tag = 0; //初始化为不分界符
    type.num = "0";
    char c1 = word;
    type.word = c1;

    if (c1 == '+') { type.tag = 1; type.num = "14"; type.word = c1; }
    else if (c1 == '-') { type.tag = 1; type.num = "15"; type.word = c1; }
    else if (c1 == '*') { type.tag = 1; type.num = "16"; type.word = c1; }
    else if (c1 == ';') { type.tag = 1; type.num = "17"; type.word = c1; }
    else if (c1 == '(') { type.tag = 1; type.num = "18"; type.word = c1; }
    else if (c1 == ')') { type.tag = 1; type.num = "19"; type.word = c1; }
    else if (c1 == '=') { type.tag = 1; type.num = "20"; type.word = c1; }
    else if (c1 == '>') { type.tag = 1; type.num = "21"; type.word = c1; }
    else if (c1 == '<') { type.tag = 1; type.num = "23"; type.word = c1; }
    else if (c1 == ':') { type.tag = 1; type.num = "26"; type.word = c1; }
    else if (c1 == '/') { type.tag = 1; type.num = "28"; type.word = c1; }
    else if (c1 == '!') { type.tag = 1; type.num = "30"; type.word = c1; }
    else if (c1 == '{') { type.tag = 1; type.num = "31"; type.word = c1; }
    else if (c1 == '}') { type.tag = 1; type.num = "32"; type.word = c1; }
    else if (c1 == '&') { type.tag = 1; type.num = "33"; type.word = c1; }
    else if (c1 == '|') { type.tag = 1; type.num = "34"; type.word = c1; }
    else if (c1 == '\n') { type.tag = 1; type.num = "35"; type.word = c1; }

    return type;
}

/*****/

```

```

Function:      // isWord
Description:   // 判断单词符号的种别码
Calls:         // 无
Input:         // 待判断的单词符号
Output:        // 无输出参数
Return:        // 返回 restype 类型变量
Others:        // tag 为 0 表示不是分界符，为 1 表示是分界符
*****/
restype isWord(string word) {
    restype type;
    type.tag = 0; // 确定不是分界符
    // 全部转化为小写
    for (int i = 0; i < word.length(); ++i) {
        if (word[i] >= 'A' && word[i] <= 'Z') {
            word[i] = word[i] + 32;
        }
    }
    // 保留字
    if (word == "void") { type.num = "1"; type.word = word; }
    else if (word == "int") { type.num = "2"; type.word = word; }
    else if (word == "float") { type.num = "3"; type.word = word; }
    else if (word == "double") { type.num = "4"; type.word = word; }
    else if (word == "if") { type.num = "5"; type.word = word; }
    else if (word == "else") { type.num = "6"; type.word = word; }
    else if (word == "for") { type.num = "7"; type.word = word; }
    else if (word == "do") { type.num = "8"; type.word = word; }
    else if (word == "while") { type.num = "9"; type.word = word; }
    else if (word == "return") { type.num = "10"; type.word = word; }
    else if (word == "break") { type.num = "11"; type.word = word; }

    else {
        int state = 1;
        for (int j = 0; j < word.length() && state == 1; j++) {
            if (word[j] == '0' || word[j] == '1' || word[j] == '2' || word[j] == '3' ||
word[j] == '4' || word[j] == '5' || word[j] == '6' ||
            word[j] == '7' || word[j] == '8' || word[j] == '9');
            else { state = 0; }
        }
        // 数字
        if (state == 1) {
            type.num = "13"; type.word = word;
        }
        // 未标识
        else {

```

```

        type.num = "12"; type.word = word;
    }
}

return type;
}

/*****
Function:      // analyse
Description:   // 词法分析主体函数
Calls:        // isDelimiter; isWord;
Input:        // 从文件中读取的原字符串
Output:       // 无输出参数
Return:       // 返回得到二元式序列组成的结果字符串
Others:       // 完成词法分析，与注释未关闭导致的错误的检测
*****/
string analyse(string my_str) {
    string result_str = ""; //记录分析结果
    string word = "";
    //获取词
    my_str = my_str + ' ';
    int length = my_str.length();
    for (int i = 0; i < length; i++) {
        if (my_str[i] != ' ' && my_str[i] != '\r\t' && my_str[i] != '\0' &&
            isDelimiter(my_str[i]).tag == 0 && (i + 1) != length) {
            word = word + my_str[i];
        }
        else { //开始分析
            if (word != "") {
                restype judge1 = isWord(word);
                result_str = result_str + "(" + judge1.num + "," + judge1.word + ")";
                word = ""; //对 word 进行清空，以便于记录下一个 word
            }
            //对当前分界符进行分析
            restype judge2 = isDelimiter(my_str[i]);
            if (judge2.tag == 1) {
                if (judge2.num == "21" && my_str[i + 1] == '=') {
                    result_str = result_str + "(41, >=)";
                    i++; //跳过双分界符的第二个字符
                }
                else if (judge2.num == "23" && my_str[i + 1] == '=') {
                    result_str = result_str + "(42, <=)";
                    i++; //跳过双分界符的第二个字符
                }
                else if (judge2.num == "23" && my_str[i + 1] == '>') {

```

```
        result_str = result_str + "(43, <> )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "26" && my_str[i + 1] == '=') {
        result_str = result_str + "(44, := )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "14" && my_str[i + 1] == '=') {
        result_str = result_str + "(45, += )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "14" && my_str[i + 1] == '+') {
        result_str = result_str + "(46, ++ )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "15" && my_str[i + 1] == '=') {
        result_str = result_str + "(47, -= )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "15" && my_str[i + 1] == '-') {
        result_str = result_str + "(48, -- )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "16" && my_str[i + 1] == '=') {
        result_str = result_str + "(49, *= )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "28" && my_str[i + 1] == '=') {
        result_str = result_str + "(50, /= )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "33" && my_str[i + 1] == '&') {
        result_str = result_str + "(51, && )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "34" && my_str[i + 1] == '|') {
        result_str = result_str + "(52, || )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "21" && my_str[i + 1] == '>') {
        result_str = result_str + "(53, >> )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "23" && my_str[i + 1] == '<') {
```

```

        result_str = result_str + "(54, << )";
        i++; //跳过双分界符的第二个字符
    }
    else if (judge2.num == "30" && my_str[i + 1] == '=') {
        result_str = result_str + "(55, != )";
        i++; //跳过双分界符的第二个字符
    }
    //*****对换行符进行处理*****
    else if (judge2.num == "35" ) {
        result_str = result_str + "\n";
    }
    //*****对注释进行处理*****
    else if (judge2.num == "28" && my_str[i + 1] == '*') {
        result_str = result_str + "(注释开始, /* )";
        i++; //跳过双分界符的第二个字符
        int state = 0;
        int j = i + 1;
        for (; j < length && state == 0; j++) {
            if (my_str[j] == '*' && my_str[j + 1] == '/') {
                state = 1;
            }
        }
        if (state == 1) { //注释有结尾
            result_str = result_str + "(注释关闭, */ )";
            i = j;
        }
        else {
            result_str = result_str + "(出现错误, 注释未关闭!)";
            i = length;
        }
    } //开始分析
    else {
        result_str = result_str + "(" + judge2.num + ", " + judge2.word + ")";
    }
} //获取词

}

return result_str;
}

/*****
Function:    // writeResult
Description: // 将分析结果写入文件
Calls:      // 无

```

```
Input:           // 结果字符串
Output:          // 结果字符串写入结果文件
*****/
void writeResult(string result_str) {
    ofstream out("output1.txt");
    if (out.is_open())
    {
        out << result_str << endl;
        out.close();
    }
}

/*****
Function:        // readTxt
Description:     // 读取文件中的字符串
*****/
string readTxt(string file)
{
    ifstream infile;
    infile.open(file.data()); //将文件流对象与文件连接起来
    assert(infile.is_open()); //若失败, 则输出错误消息, 并终止程序运行

    if (!infile) {
        cerr << "读取文件出错了!" << endl;
        exit(1);
    }

    char ch;
    string my_str = "";
    infile >> noskipws;
    while (infile.get(ch))//从文件读取字符进来
    {
        my_str += ch;
    }
    infile.close(); //关闭文件输入流
    return my_str;
}

int main() {
    string my_str = readTxt("input1.txt");
    string result_str = analyse(my_str);
    writeResult(result_str);
    return 0;
}
```