**词法分析程序测试文档**

**一、测试输入**

输入分为两个文件，**init.txt提供所有关键字集**，程序初始化时将关键字集读入到关键字表中为后续查阅做准备，用**文件形式存储**便于添加和删除。

其内容为：

int

char

for

switch

string

case

return

enum

union

cout

double

if

else

while

main

void

default

如果有需要，在init中直接添加相应的关键字即可。

**in.txt为测试用例**，选取了一段有错误的C程序作为测试。

测试用例的内容为：

double add(double \_a,double \_b)

{

return \_a + \_b;

}

void myprint\_hello\_world()

{

%printf("hello,world");

}

/\*

\* name : laoshi

\* date : 2016.10.11

\* task : lexical test

\* comment : i love p.c.

\*/

int main(void)

{

double sum = 0;

int cnt = 0;

int nums[100] = {0};

char ch = 's';

char ch\_enter = '\n';

for(int i = 1; i <= MAX\_TEST ;i ++)

{

add(sum,1.45E-2);

}

// some test for arith operators

if(sum == 100) cnt += 2;

else if(su$m > 100) cnt -= 2;

else cnt --;

cnt = 0;

while(cnt < 100)

nums[cnt] += 1;

myprint\_hello\_world();

return 0;

}

**二、运行结果**

< KEY , double >

< ID , add >

< SINGLE\_OP , ( >

< KEY , double >

< ID , \_a >

< SINGLE\_OP , , >

< KEY , double >

< ID , \_b >

< SINGLE\_OP , ) >

< SINGLE\_OP , { >

< KEY , return >

< ID , \_a >

< AIRTH\_OP , + >

< ID , \_b >

< SINGLE\_OP , ; >

< SINGLE\_OP , } >

< KEY , void >

< ID , myprint\_hello\_world >

< SINGLE\_OP , ( >

< SINGLE\_OP , ) >

< SINGLE\_OP , { >

ERROR : Lexical error in line 8

< ID , printf >

< SINGLE\_OP , ( >

< LITERAL , "hello,world" >

< SINGLE\_OP , ) >

< SINGLE\_OP , ; >

< SINGLE\_OP , } >

< KEY , int >

< KEY , main >

< SINGLE\_OP , ( >

< KEY , void >

< SINGLE\_OP , ) >

< SINGLE\_OP , { >

< KEY , double >

< ID , sum >

< ASSIGN\_OP , = >

< NUM , 0 >

< SINGLE\_OP , ; >

< KEY , int >

< ID , cnt >

< ASSIGN\_OP , = >

< NUM , 0 >

< SINGLE\_OP , ; >

< KEY , int >

< ID , nums >

< SINGLE\_OP , [ >

< NUM , 100 >

< SINGLE\_OP , ] >

< ASSIGN\_OP , = >

< SINGLE\_OP , { >

< NUM , 0 >

< SINGLE\_OP , } >

< SINGLE\_OP , ; >

< KEY , char >

< ID , ch >

< ASSIGN\_OP , = >

< CHAR , 's' >

< SINGLE\_OP , ; >

< KEY , char >

< ID , ch\_enter >

< ASSIGN\_OP , = >

< CHAR , 'ESC' >

< SINGLE\_OP , ; >

< KEY , for >

< SINGLE\_OP , ( >

< KEY , int >

< ID , i >

< ASSIGN\_OP , = >

< NUM , 1 >

< SINGLE\_OP , ; >

< ID , i >

< REALATION\_OP , <= >

< ID , MAX\_TEST >

< SINGLE\_OP , ; >

< ID , i >

< AIRTH\_OP , ++ >

< SINGLE\_OP , ) >

< SINGLE\_OP , { >

< ID , add >

< SINGLE\_OP , ( >

< ID , sum >

< SINGLE\_OP , , >

< NUM , 0.0145 >

< SINGLE\_OP , ) >

< SINGLE\_OP , ; >

< SINGLE\_OP , } >

< KEY , if >

< SINGLE\_OP , ( >

< ID , sum >

< REALATION\_OP , == >

< NUM , 100 >

< SINGLE\_OP , ) >

< ID , cnt >

< ASSIGN\_OP , += >

< NUM , 2 >

< SINGLE\_OP , ; >

< KEY , else >

< KEY , if >

< SINGLE\_OP , ( >

< ID , su >

ERROR : Lexical error in line 31

< ID , m >

< REALATION\_OP , > >

< NUM , 100 >

< SINGLE\_OP , ) >

< ID , cnt >

< ASSIGN\_OP , -= >

< NUM , 2 >

< SINGLE\_OP , ; >

< KEY , else >

< ID , cnt >

< AIRTH\_OP , -- >

< SINGLE\_OP , ; >

< ID , cnt >

< ASSIGN\_OP , = >

< NUM , 0 >

< SINGLE\_OP , ; >

< KEY , while >

< SINGLE\_OP , ( >

< ID , cnt >

< REALATION\_OP , < >

< NUM , 100 >

< SINGLE\_OP , ) >

< ID , nums >

< SINGLE\_OP , [ >

< ID , cnt >

< SINGLE\_OP , ] >

< ASSIGN\_OP , += >

< NUM , 1 >

< SINGLE\_OP , ; >

< ID , myprint\_hello\_world >

< SINGLE\_OP , ( >

< SINGLE\_OP , ) >

< SINGLE\_OP , ; >

< KEY , return >

< NUM , 0 >

< SINGLE\_OP , ; >

< SINGLE\_OP , } >

---Token Statistic---

total char : 631

KEY : 21

ID : 29

NUM : 14

LITERAL : 1

CHAR : 2

SINGLE\_OP : 53

ASSIGN\_OP : 10

LOGIC\_OP : 0

RELATION\_OP : 4

ARITH\_OP : 3

BITWISE\_OP : 0

**三、结果分析说明**

参照设计文档，每个token以<类别，属性>的方式输出，输出结果和统计结果均是正确的。程序可以正确处理//和/\*注释，这反映在注释中的内容被完全跳过，没有产生任何token。测试用例中设计了两个词法错误，分别是

%printf("hello,world"); 和

else if(su$m > 100) cnt -= 2;

这两个词法错误均被检测出来并输出了相应的行号

ERROR : Lexical error in line 8

ERROR : Lexical error in line 31

对于一些重要token的解释：

**1、< ID , printf >**

可以看出任何函数都被当作ID处理，即使是库函数（这里并没有体现）也不例外。而且最重要的是，紧邻它的词法分析错误由于有恐慌模式，并没有影响到printf这个标识符的输出。

**2、< NUM , 0.0145 >**

可以看出原本的1.45E-2被转化成了小数的形式，这步处理在词法分析程序中已经完成了，为语法分析程序节约了处理时间。

**3、< CHAR , 'ESC' >**

由于C++没有原生字符串的输出函数，这里想直接输出转义字符，转义字符均用ESC表示，具体的属性储存在token中，不予以打印。

**4、< ID , su >ERROR : Lexical error in line 31< ID , m >**

读入su之后由于遇到非法字符，系统先输出su为标识符，并将非法字符跳过，直到识别了另一个词法单元，也即剩余的m，被识别为另一个标识符。

**5、---Token Statistic---**

这之后是统计结果，按照设计文档中列出的所有属性，程序对它们进行了统计，核对后数值是正确的。