需求分析

1. 对C语言进行词法分析，并以记号形式输出
2. 可识别注释
3. 可统计行数、字数、单词数（标点和空格不算），输出统计结果
4. 检查源程序中的错误并定位
5. 检错后，进行适当恢复，使程序继续进行，通过一次词法分析处理，检查程序所有错误并报告。

C语言有标识符、关键字、分界符、运算符、常量、注释六类

关键字有32个

标识符第一个字符由字母或者下划线，剩下的由字母或者数字或者下划线构成

运算符也有常用的几十个

常量有整数、浮点数、字符、字符串常量

注释有//和/\*\*/两种注释

概要设计

运行环境：Microsoft Windows 10 1803

开发环境：Microsoft Visual Studio 2017 Community

一、版块及其调用关系

1. ERROR.h和ERROR.cpp

头文件内存储ERROR类的声明，也存储错误代码

1. INPUT\_BUFFER.h和INPUT\_BUFFER.cpp

缓冲区INPUT\_BUFFER类，用于读入输入的源代码文件，有两个缓冲区buffer1和buffer2，每个缓冲区大小为4KB+1B，末尾设置为岗哨，提高效率。

1. lexer.h和lexer.cpp

词法分析器的主要程序，包含scan和reverse两个程序，reverse为识别保留字程序，被scan调用。

1. TOKEN.h

TOKEN类的声明和一些赋值函数，TOKEN作为词素识别后的输出形式存在

1. main.cpp

程序启动（命令行方式下的参数设定），测试代码（调用lexer输出到文件中）

调用关系：

lexer调用TOKEN/INPUT\_BUFFER/ERROR

INPUT\_BUFFER调用ERROR/TOKEN

Main仅调用lexer的scan程序和某些测试代码

二、全局变量/常量/宏定义及其说明

全局变量：

INPUT\_BUFFER buffer

实例化INPUT\_BUFFER，作为输入缓冲区，整个程序仅此一个实例

ERROR lexerError

实例化ERROR，作为错误检查类，其相关数据和函数皆用于错误检查和统计，整个 程序仅此一个实例

bool isEOF

布尔变量输入文件是否到达结尾。

常量：

无

宏定义：

TOKEN类中：皆用于定义TOKEN第一分量TAG值

//运算符TAG定义

#define TAG\_POINTER 256 //->

#define TAG\_SELFADD 257 //++

#define TAG\_SELFSUB 258 //--

#define TAG\_SIZEOF 259 //sizeof

#define TAG\_LEFTSHIFT 260 //<<

#define TAG\_RIGHTSHIFT 261 //>>

#define TAG\_GE 262 //>=

#define TAG\_LE 263 //<=

#define TAG\_EQ 264 //==

#define TAG\_NEQ 265 //!=

#define TAG\_AND 266 //&&

#define TAG\_OR 267 //||

#define TAG\_DIV\_ASSIGN 268 // /=

#define TAG\_MUL\_ASSIGN 269 //\*=

#define TAG\_MOD\_ASSIGN 270 //%=

#define TAG\_ADD\_ASSIGN 271 //+=

#define TAG\_SUB\_ASSIGN 272 //-=

#define TAG\_LEFTSHIFT\_ASSIGN 273 //<<=

#define TAG\_RIGHTSHIFT\_ASSIGN 274 //>>=

#define TAG\_BITAND\_ASSIGN 275 //&=

#define TAG\_BITNOR\_ASSIGN 276 //^=

#define TAG\_BITOR\_ASSIGN 277 //|=

//常量TAG定义

#define TAG\_INTEGER 278

#define TAG\_REAL 279

#define TAG\_CH 281

#define TAG\_STR 282

//标识符TAG定义

#define TAG\_ID 283

//保留字TAG定义

#define TAG\_IF 284

#define TAG\_ELSE 285

#define TAG\_WHILE 286

#define TAG\_DO 287

#define TAG\_CLASS 288

#define TAG\_BREAK 289

#define TAG\_AUTO 290

#define TAG\_CASE 291

#define TAG\_CHAR 292

#define TAG\_CONST 293

#define TAG\_CONTINUE 294

#define TAG\_DEFAULT 295

#define TAG\_DOUBLE 296

#define TAG\_ENUM 297

#define TAG\_EXTERN 298

#define TAG\_FLOAT 299

#define TAG\_FOR 300

#define TAG\_GOTO 301

#define TAG\_INT 302

#define TAG\_LONG 303

#define TAG\_REGISTER 304

#define TAG\_RETURN 305

#define TAG\_SHORT 306

#define TAG\_SIGNED 307

#define TAG\_STATIC 308

#define TAG\_STRUCT 309

#define TAG\_SWITCH 310

#define TAG\_TYPEDEF 311

#define TAG\_UNION 312

#define TAG\_UNSIGNED 313

#define TAG\_VOID 314

#define TAG\_VOLATILE 315

//失效的TOKEN应该舍去

#define TAG\_NULL -1

ERROR类中：

#define L001 1

#define L002 2

#define L003 3

#define L004 4

#define L005 5

#define L006 6

#define L007 7

#define R001 1001

#define R002 1002

L001表示\\*\*\类注释没有完成匹配程序就已结束

L002表示输入了非法字符，状态机无法识别

L003表示识别NUM时输入小数点后没有再输入别的数字，如123.

L004表示识别NUM时输入E（+、-）后没有再输入别的数字，如3E、2E-、4E+

L005表示输入了’’

L006表示输入了\x等无法识别的十六进制转移字符

L007表示输入了\等无法识别的转义字符

R001作为内部报错代码，用于放在一些跳转不到的分支报错

R002内部报错代码，表示提交的错误代码未定义

INPUT\_BUFFER类中：

#define INPUT\_BUFFER\_SIZE (1<<15) + 1 //输入缓冲区大小

三、接口设计

用户接口：

使用命令行模式，参数可选0、1、2个：0个参数情况下，程序直接读取同一目录下的text.txt作为输入，以outcome.txt作为结果输出；1个参数情况下，程序以输入的参数作为输入文件所在路径，输出到程序同一目录下的outcome.txt中；2个参数情况下，程序以第一个参数作为输入文件的所在路径，以第二个参数作为输出文件的所在路径。若编译成功，程序会将行数、词数、字数等统计信息输出到命令行中；若未成功，则会将错误代码、错误提示、错误位置输出到命令行中。

内部接口：

scan作为整个词法分析器程序的接口，调用函数会直接得到输入的下一个词素TOKEN结果。ERROR.operator(int errorcode)作为报错程序，提供错误提醒接口。INPUT\_BUFFER.get\_char(char& ch)作为得到下一个字符函数，把结果赋值到ch中供scan调用，也提供INPUT\_BUFFER.retract()函数，回退一个字符（往前读一个字符是为了确定词素）。

四、数据类型及函数作用

1、类TOKEN

变量：

int tag TOKEN二元组第一分量

以下四个均根据TAG启用其中一个或者一个也不启用作为第二分量

long long int value\_int TAG为INTEGER时启用，存储整数数值

double value\_float TAG为REAL时启用，存储浮点数值

int value\_char TAG为CH时启用，存储字符常量值

string lexeme TAG为ID或者STR常量时启用，存储ID名或字符串常量值

当TAG为运算符、保留字、分界符等不启用第二分量，直接用第一分量TAG识别对应词素。

函数：

void num(value)有三个重载，将三个不同类型的value值赋值给value\_int/float/char值，void word(string lexeme)将lexeme赋值给this->lexeme，TOKEN（int tag）初始化函数，将tag赋值给this->tag

2、类INPUT\_BUFFER

变量：

char buffer1[INPUT\_BUFFER\_SIZE] 缓冲区1

char buffer1[INPUT\_BUFFER\_SIZE] 缓冲区2

bool isbuffer1 指明当前使用的缓冲区

bool isRetract 如果回退跨越了缓冲区则置1防止再次读取

char\* fileName 输入文件路径

char\* outfileName 输出文件路径

int begin 词素开始指针

int forward 词素向前指针

函数：

void retract(void) 让forward指针回退

void dealEOF(void) 处理读取字符过程中读到的EOF标志

void get\_char(char& ch) 读取字符并存在ch中传给scan函数

3、类ERROR

变量：

int lineIterator 行数迭代器

int wordIterator 字符迭代器

int lexmeIterator 统计共多少个词素

int sumword 统计共多少个字符

函数：

void operator()(int errorcode) 根据提供的错误代码进行错误提醒并退出程序

4、lexer.cpp中的scan和reverse函数

TOKEN\* lexer\_scan(void) 每次调用返回下一个分析到的词素TOKEN，若读到结尾，isEOF会被置为true

void lexer\_scan\_reverse(TOKEN\* t)根据传入的TOKEN->lexeme进行匹配判断是否为关键字，若是关键字则将TOKEN->TAG改为相应关键字。因为lexer\_scan中将关键字和标识符的识别放在一起，所以单独写一个程序用于区分关键字和标识符，仅被lexer\_scan调用。

5.main和printTOKEN

两个函数都作为测试函数

void printTOKEN(TOKEN\* t) 根据TOKEN.tag和第二分量，将TOKEN输出为便于识别的二元组以测试代码结果是否正确

main根据传入的命令行参数判断处理模式（输入文件路径和输出文件路径），然后调用scan并输出结果

五、尚未解决的问题及可能的扩展方向

1、数字常量的识别未限定范围，数字过大或精度问题会导致溢出

2、reverse函数的判定有些重复，可以进一步提高效率

3、scan函数过长，应该拆分成各个部分以提高可读性

4、没有提供词法分析器到符号表的API

详细设计

1. TOKEN

class TOKEN

{

public:

int tag;

long long int value\_int;//识别出INTEGER后存储在该变量中

double value\_float;//识别出REAL后存储在该变量中

int value\_char;//识别出CH后存储在该变量中

string lexeme;//识别出标识符(ID)或STR后赋值lexeme

TOKEN(int tag) { this->tag = tag; }

void num(long long int value) { this->value\_int = value; }

void num(double value) { this->value\_float = value; }

void num(int value) { this->value\_char = value; }

void word(string lexeme) { this->lexeme = lexeme; }

};

简单明了，函数仅赋值，不做赘述。

1. INPUT\_BUFFER

class INPUT\_BUFFER

{

public:

char buffer1[INPUT\_BUFFER\_SIZE];

char buffer2[INPUT\_BUFFER\_SIZE];

int begin = INPUT\_BUFFER\_SIZE - 1;//开始指针

int forward = INPUT\_BUFFER\_SIZE - 1;//向前指针

char\* fileName = nullptr;

char\* outfileName = nullptr;

bool isbuffer1 = false;

bool isRetract = false;//若回退跨越两个缓冲区，则置true,下一次跨越时不再读取文件

INPUT\_BUFFER();

void retract(void);

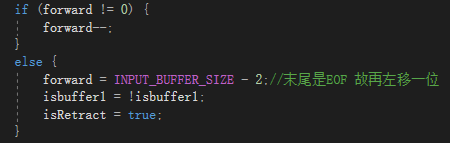
void dealEOF(void);//EOF处理

void get\_char(char& ch);

};

void retract(void)

如果forward在开头（即为0），表示刚刚跨越缓冲区，此时缓冲区已经刷新过，回退后再跨越时不应该刷新，所以forward被置为上一个缓冲区最后一个（INPUT\_BUFFER\_SIZE-2,-1为末尾岗哨位EOF），且isbuffer1应该翻转（跨越缓冲区）isRetract=true，防止下一次刷新缓冲区

代码也较为简单

void dealEOF(void)

调用这个函数时，必定读到了EOF，处理逻辑是，判断forward是否在末尾，若是则跨越缓冲区（根据isRetract判断该不该刷新下一个缓冲区），若不是则不做任何处理（调用这个函数的函数会根据forward位置再一次判定是否为末尾）

void get\_char(char& ch)

读取下一个字符，读到EOF就调用dealEOF，调用完判定是否末尾，不是就将现在的字符传给ch，是就传’\n’，以结束词法分析器现在的所有动作，并将isEOF置为true，当ch读到换行符且不是结尾时，ERROR类的行、字统计指针都进行更新，否则只更新字统计指针。

1. ERROR

class ERROR

{

public:

int lineIterator = 1;

int wordIterator = 1;

int sumword = 0;

int lexmeIterator = 0;//单词个数(标点和空格不计数)

void operator()(int errorcode);

};

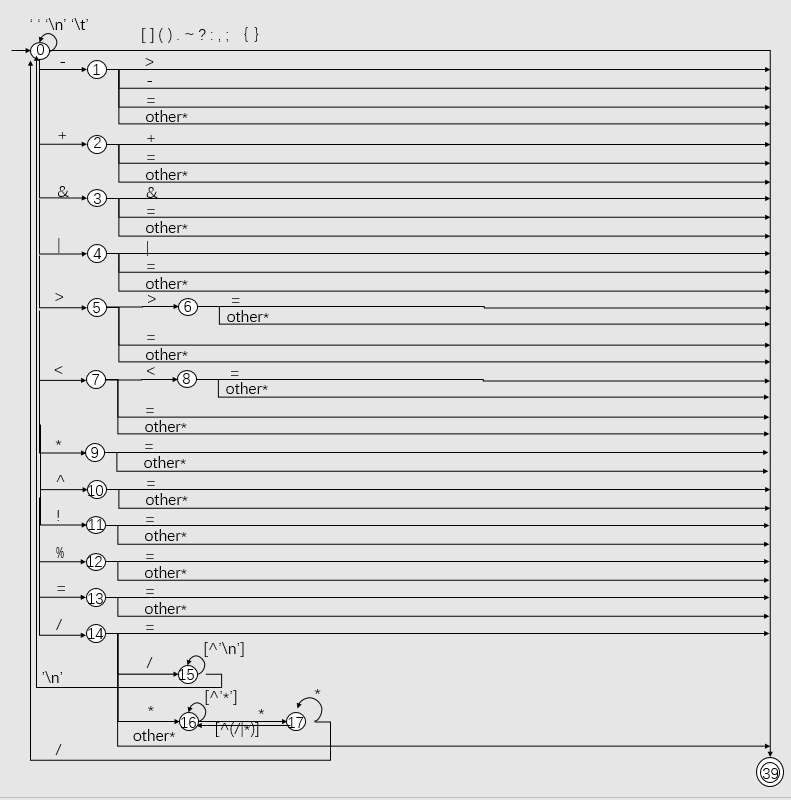
void operator()(int errorcode)

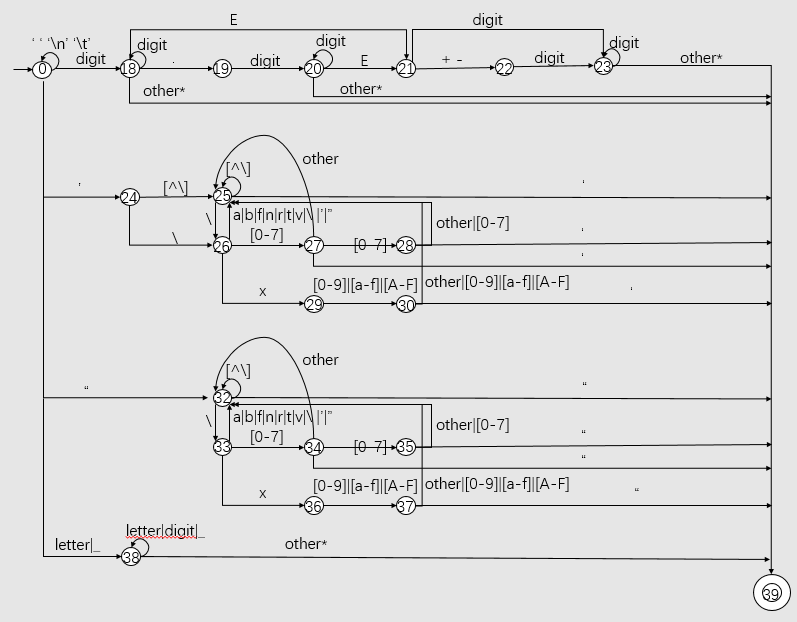
根据参数值进行错误提醒，提醒内容在概要设计的宏定义栏中

1. lexer

TOKEN\* lexer\_scan(void);

void lexer\_scan\_reverse(TOKEN\* t);

TOKEN\* lexer\_scan（void）进行词法分析，标识符、关键字、常量、运算符、分界符、注释全都在同一自动机中识别。并将结果作为TOKEN返回，自动机状态图如下：

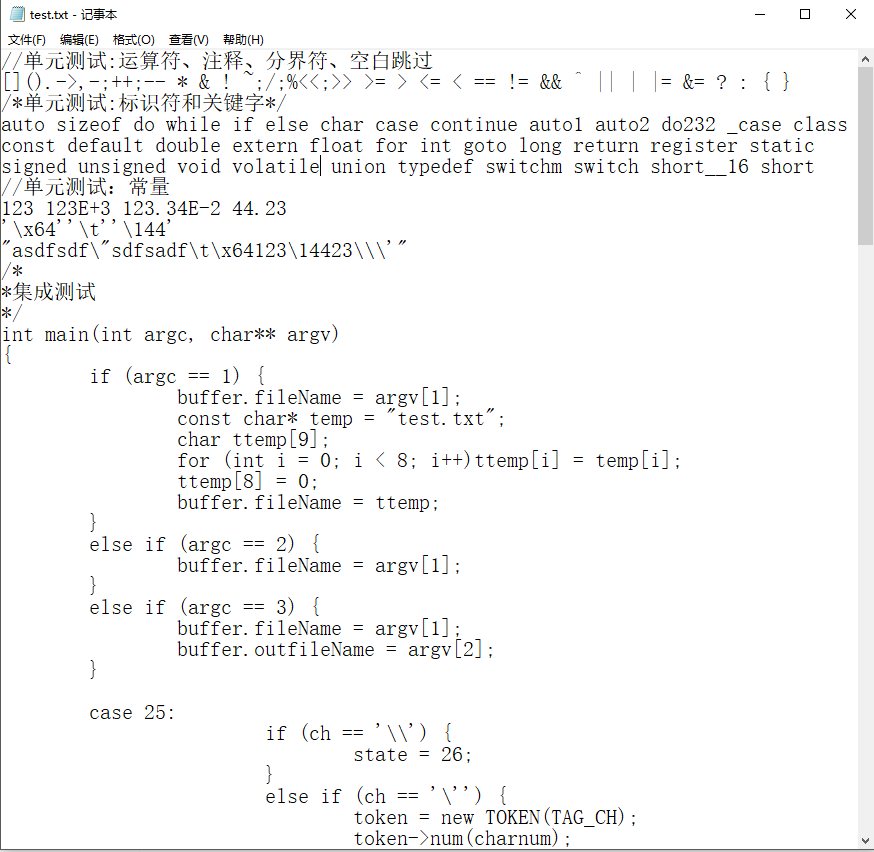


所有传入字符带\*表示需要回退，即调用INPUT\_BUFFER.retract()

void lexer\_scan\_reverse(TOKEN\* t)

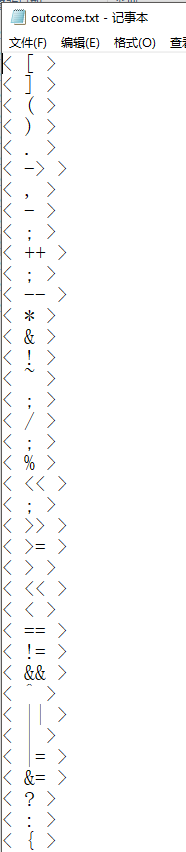
和scan类似，进行最简单的字符串匹配，与32个保留字字符串匹配，若匹配上则将TAG换成对应的保留字TAG，如果一个都没匹配上，则表示这个是标识符，不做任何处理。

测试结果

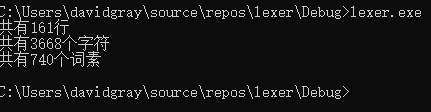


输入源程序

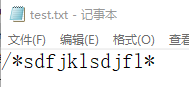
输出结果



由于过长，不一一展示，输入文件和输出文件都在附件中

命令行中提示

错误测试：

输入：

输出：