编译器专题实验报告

**实验一：词法分析（独立模式）**

**一、实验目的**

构建词法分析程序能将源语言程序作为输入，并输出词法记号串到文件中。

包含功能：

（1）单词设计包含主文法中所有词法单位；

（2）一遍扫描或含预处理遍，能删除注解，并允许空白字符串作为分隔；

（3）分析框架scanner()允许连续调用直到输入串被扫描完毕，每调用一次返回一个记号；

（4）有联合DFA设计结果以及超前搜索功能；

（5）（可选）词法错误处理。

编写某语言的词法分析器的源程序并调试通过。其中词法分析程序既可以自己手动去完成，也可以利用LEX自动生成。可使用win-flex工具完成上述语言的正则式，实现词法分析。

**二、实验内容（必做）**

**1.实验要求**

根据给定的文法设计并实现词法分析程序，从源程序中识别出单词，记录其单词类别和单词值，输入输出及处理要求如下：

（1）单词类别码 单词的字符/字符串形式(中间仅用一个空格间隔)。

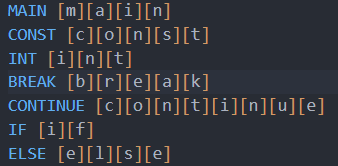
（2）单词的类别码统一按如下形式定义：

| **单词名称** | **类别码** | **单词名称** | **类别码** | **单词名称** | **类别码** | **单词名称** | **类别码** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Ident** | IDENFR | ! | NOT | \* | MULT | = | ASSIGN |
| **IntConst** | INTCON | && | AND | / | DIV | ; | SEMICN |
| **FormatString** | STRCON | || | OR | % | MOD | , | COMMA |
| main | MAINTK | while | WHILETK | < | LSS | ( | LPARENT |
| const | CONSTTK | getint | GETINTTK | <= | LEQ | ) | RPARENT |
| int | INTTK | printf | PRINTFTK | > | GRE | [ | LBRACK |
| break | BREAKTK | return | RETURNTK | >= | GEQ | ] | RBRACK |
| continue | CONTINUETK | + | PLUS | == | EQL | { | LBRACE |
| if | IFTK | - | MINU | != | NEQ | } | RBRACE |
| else | ELSETK | void | VOIDTK |  |  |  |  |

**2.正则表达式的编写**

实验中主要使用lex工具将源程序转为词法分析器，需要使用正则表达式进行字符串的匹配，实验中用到的部分正则表达式如下所示：

（1）保留字：部分用到的保留字的正则表达式如下所示：



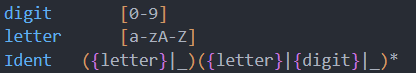
对于保留字编写正则表达式比较简单，直接编写即可。

（2）数字：本实验中定义了整型数字和浮点数字（可以使用科学计数法表示），如下所示：



其中digit定义为[0-9]，整型数字就是各种数字重复一到多次，浮点数则是以数字开头，小数点后为多个数字或者使用含有e或E的科学计数法表示。

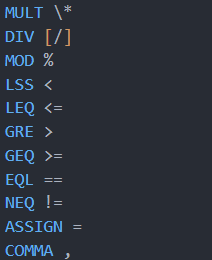
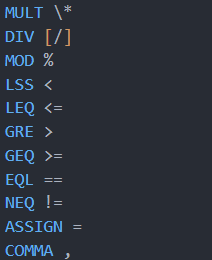
（3）ID：以字母或者下划线开头，后面可以含有数字、字母和下划线：



（4）字符串：以引号开头，引号结尾，中间匹配除了引号之外的字符



（5）数值，比较运算符和分隔符：直接定义即可，部分如下所示：

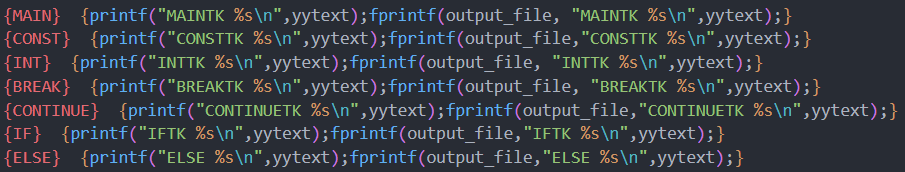
 

（6）间隔符：这里包括空格、\r和\t，这里没有把\n放在一起因为后续要针对\n进行行数统计，而针对空格等运算符不需要进行操作。



**3. 规则编写**

（1）正常输出：在此实验中，当匹配到对应的字符串时，进行输出到cmd和文件中的操作，所以主要的处理规则类似下图所示：



（2）换行处理：后续错误处理需要将对应的错误定位到对应的行，所以需要定义一个变量nline进行行数的统计，读入\n时统计的行数加一：



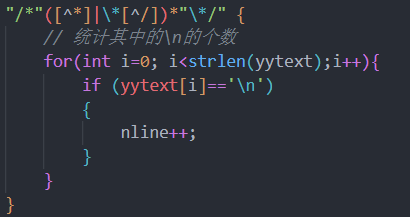
（3）头文件处理：当读入以#开头的一行时（不匹配\n以后的内容），说明引用了头文件，需要输出其引用的内容：



（3）注释处理：C语言的注释分为两种，首先考虑//的注释，一行中从//开始直到\n均为注释内容，因为读入换行了，因此行数统计需要加一：

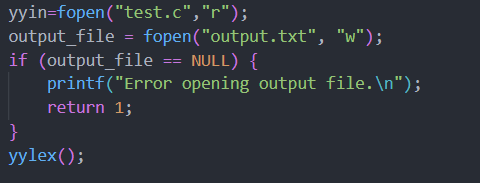


其次考虑/\*\*/的注释，这种注释可以使用多行，其匹配的正则表达式可以按如下书写："/\*"([^\*]|\\*[^/])\*"\\*/"，表示匹配以/\*开头，以\*/结尾的注释内容。其中[^\*]表示匹配除了\*之外的任意字符，\\*[^/]表示匹配\*后面紧跟着除了/之外的任意字符。整个表达式的作用是匹配/\*和\*/之间的任意字符，包括换行符。因为匹配了换行符，所以对于此类注释需要进行换行符的统计并加到nline中：



**4.主函数编写**

主函数中需要开启需要输出的文件，并进行词法分析匹配，如下所示：



**5.实验结果分析**

本实验使用简单的C语言程序进行验证，程序代码中包含注释，代码如下所示：

const int array[2] = {1,2};

int main(){

int c;

int b = 1;

/\*

123456

\*/

c = getint();

printf("output is %d",c);

if (c > 1 || b>1)

{

printf("output is %d",c);

}

if (c > 1 && b>1)

{

printf("output is %d",c);

}//123456

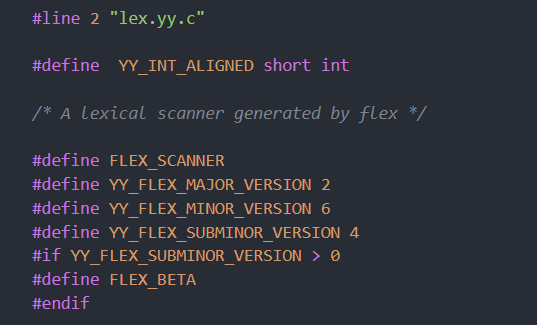
return c;

}

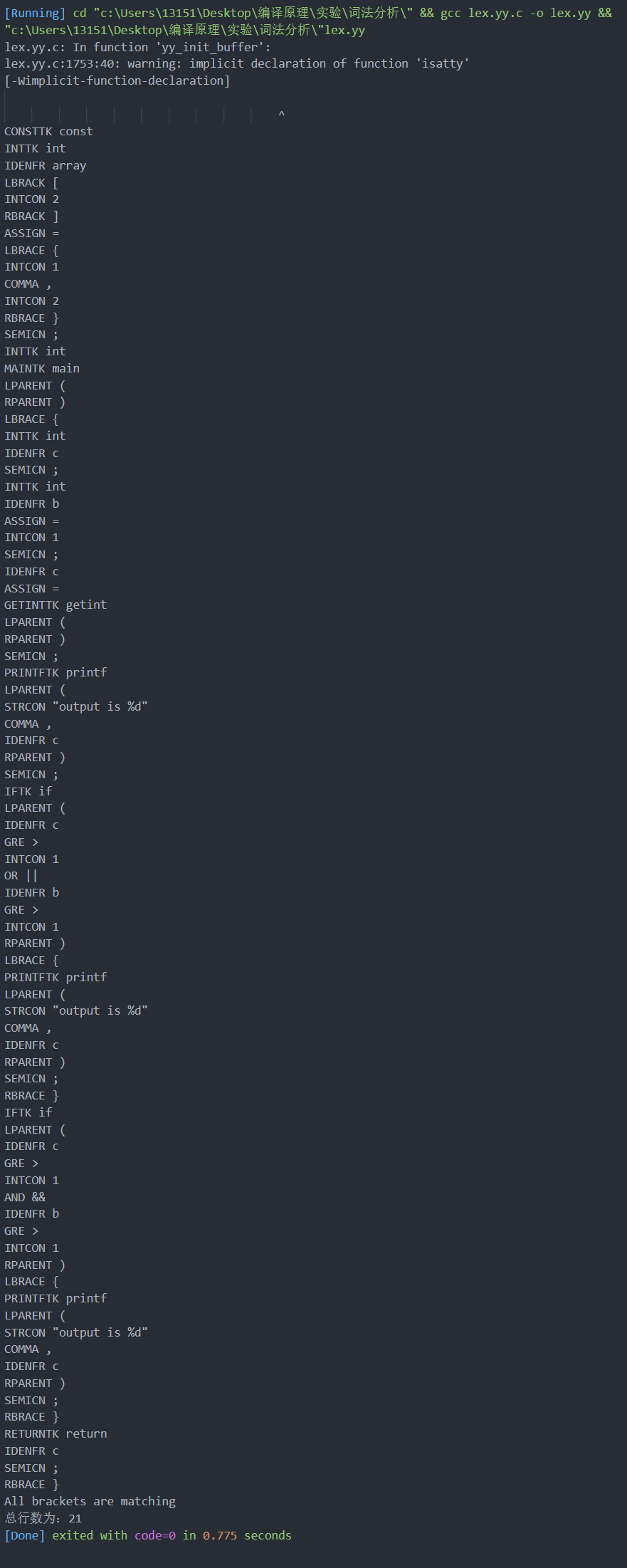
使用win-flex工具可以将编写的test.l文件转换为对应的lex.yy.c文件，而后运行该C语言文件即可得到词法分析结果，下图使用了win-flex工具：



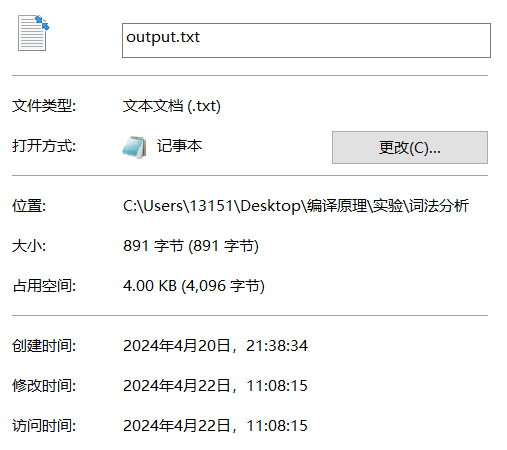
生成的lex.yy.c文件部分内容如下所示，该文件中也包含了上述翻译规则：



运行该文件后可以得到上述程序的词法翻译结果：



同时，output.txt中也会输出相同的内容，这里不重复放了，这里用output.txt的修改时间作为实验的完成时间。



**三、实验内容（选做）**

**1. 错误检查：**当检查到无法匹配的字符时需要报错并返回无法匹配到的字符的行号，首先定义无法匹配的符号other，表示可以匹配任何字符

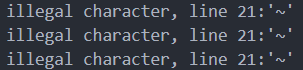


然后在规则最后加上无法匹配时的报错处理：



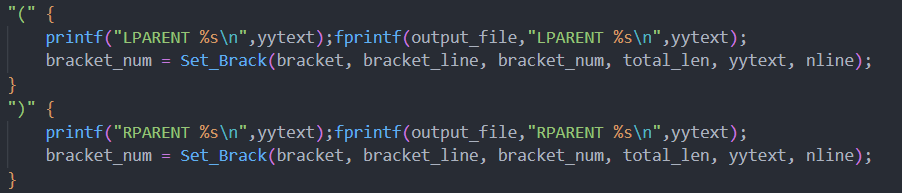
实验结果：

在原来的C语言测试文件最后一行输入无法识别的~~~串，则会输出以下内容，表示现有的规则无法匹配文本中的内容。

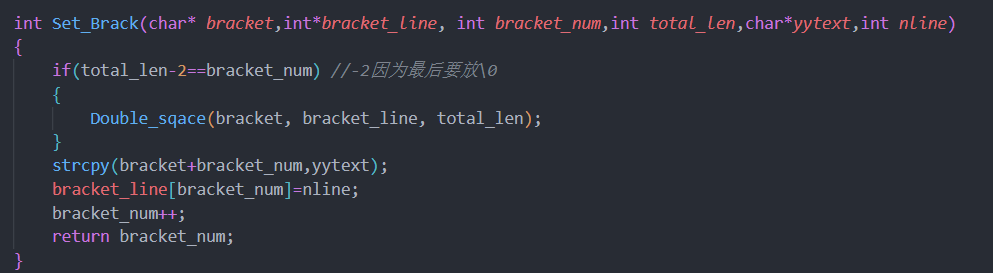


**2. 括号匹配检查：**在lex文件中还可以实现括号匹配的功能，实现的主要思路为，在词法分析时先统计出现的所有括号到数组bracket中并记录对应括号出现的行号，在分析完成之后对数组bracket中的括号进行匹配检查，使用栈的数据结构实现，最后返回哪一行的哪个括号没有匹配上。

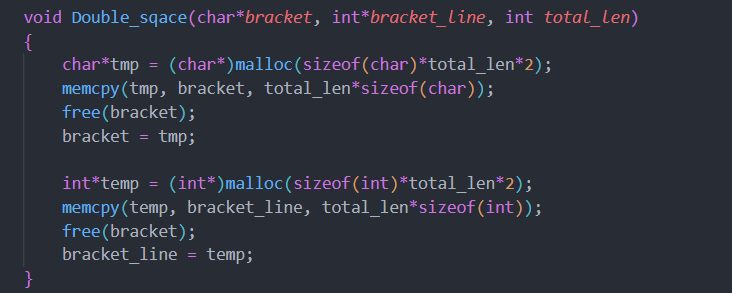
（1）规则编写：当读入一个括号（小括号/中括号/大括号）时，将其放入数组：



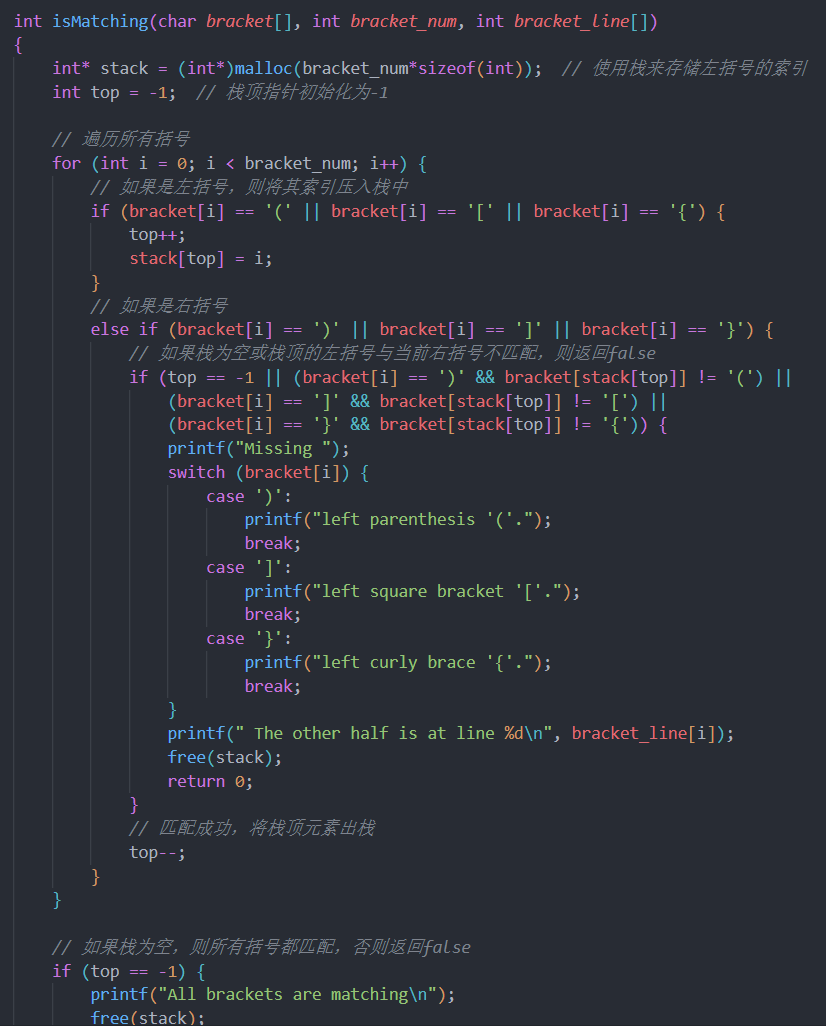
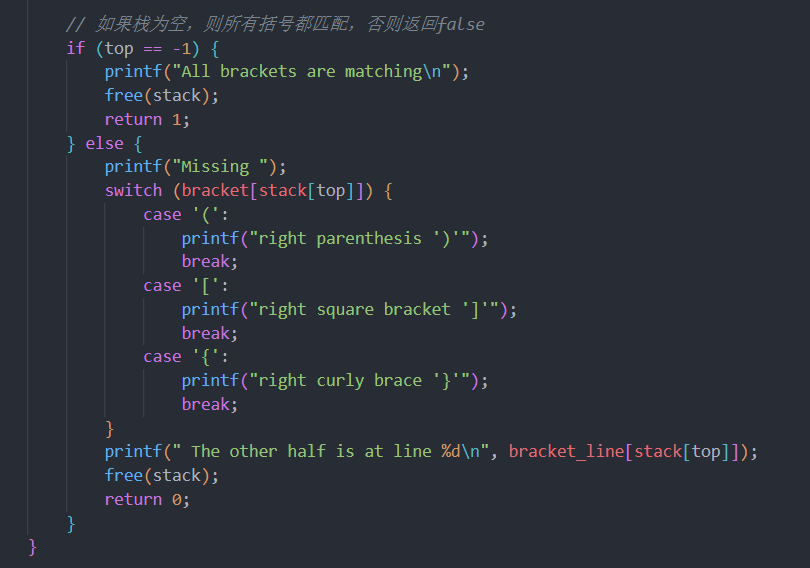
其中，bracket\_num记录记录了括号的总个数，bracket数组记录括号，bracket\_line记录对应的行数，Set\_Brack函数如下：



考虑到数组大小固定可能无法记录所有的括号，这里使用动态分配内存，当数组大小不够时将大小翻倍：



在记录完所有括号之后，在主函数中调用isMatching函数进行括号匹配的判断，其中使用栈的结构来存储左括号，如果发现栈顶的左括号和读到的右括号能够匹配则说括号匹配成功，否则右括号缺少对应的左括号，如果读完所有右括号栈中还有左括号，则说明缺少右括号：

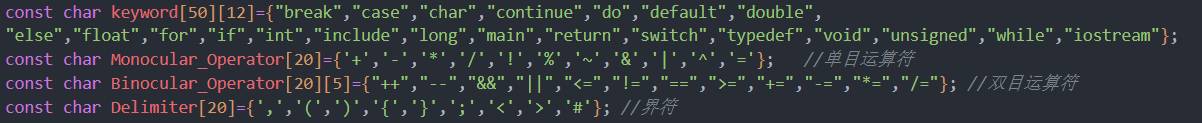
实验结果：删去测试代码最后的一个大括号，运行结果中程序发现了最后的一个大括号缺少了，对应的左括号出现在第三行。



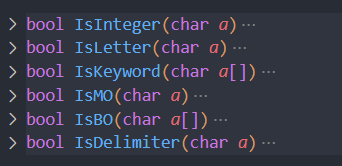
**3. 尝试自己写scanner**

以上的词法分析均由lex自动实现，但是我们也可以自己实现一部分词法分析器的功能，下面我们以识别一些简单字符为例。

我们自行设计的scanner仅识别以下简单的关键字、单目双目运算符和分界符：



首先，为了判断字符串是否属于以上的符号，我们需要定义用于判断字符串是否属于以上符号的函数：



而后，我们需要定义扫描函数，对其输入文本中的内容进行扫描（具体代码见附件），需要注意的是，当扫描到字符时，我们要进行超前搜索，判断读入的字符后面是否紧接着字符，直到读入完整的字符串，判断是否为关键字：



双目运算符和单目运算符也同理。

使用如下简单程序进行检验：

#include<iostream>

int main()

{

int aa;

while(1)

{

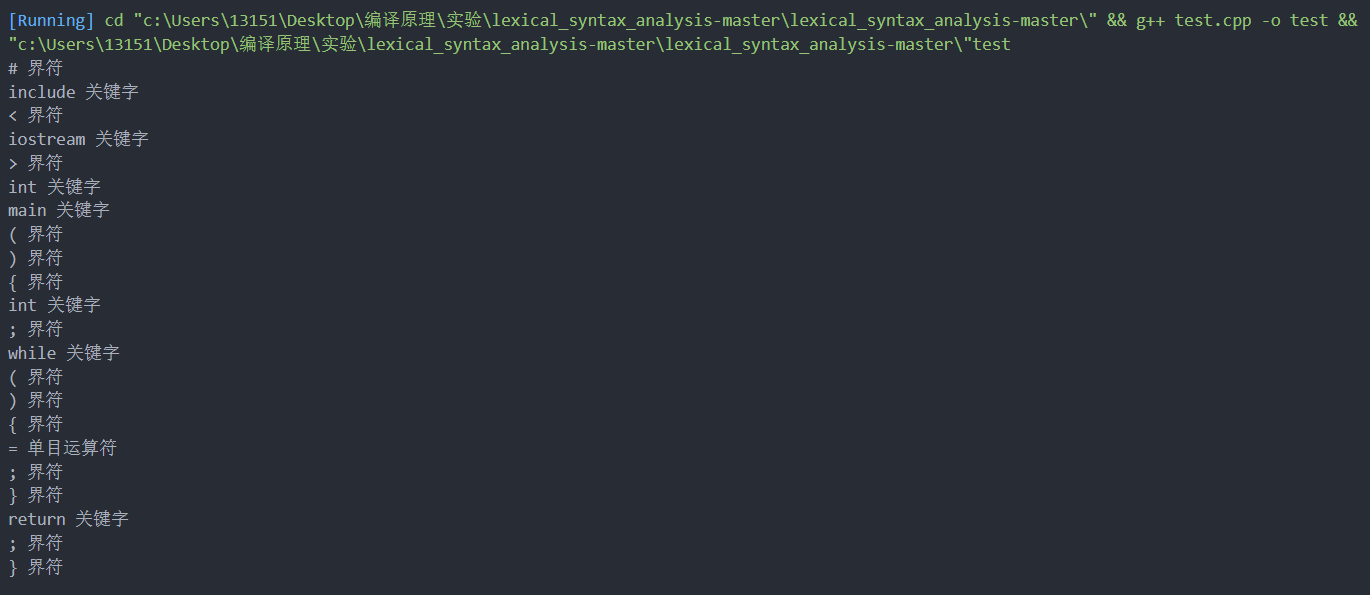
aa=1;

}

return 1;

}

运行后发现scanner可以识别出其中的我们定义的关键字等。这里只识别了我们定义的关键字，考虑到词法分析后面还有课，后面可能尝试从构建NFA开始，建立可以识别数字、ID等的scanner。



**四、遇到的问题**

1.一开始不太清楚如何使用flex，通过在Bilibili寻找视频讲解了解了如何使用win-flex。

2.实验中发现最开始的声明部分不能调用函数，一开始在其中使用malloc开辟bracket数组失败了，查阅CSDN发现不能调用函数。

**五、附件：代码**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int bracket\_num = 0;

int total\_len = 100;

char\* bracket = NULL; // 不允许函数调用

int\* bracket\_line = NULL;

FILE\* output\_file;

int nline = 1;

void Double\_sqace(char\*bracket, int\*bracket\_line, int total\_len)

{

char\*tmp = (char\*)malloc(sizeof(char)\*total\_len\*2);

memcpy(tmp, bracket, total\_len\*sizeof(char));

free(bracket);

bracket = tmp;

int\*temp = (int\*)malloc(sizeof(int)\*total\_len\*2);

memcpy(temp, bracket\_line, total\_len\*sizeof(int));

free(bracket);

bracket\_line = temp;

}

int Set\_Brack(char\* bracket,int\*bracket\_line, int bracket\_num,int total\_len,char\*yytext,int nline)

{

if(total\_len-2==bracket\_num) //-2因为最后要放\0

{

Double\_sqace(bracket, bracket\_line, total\_len);

}

strcpy(bracket+bracket\_num,yytext);

bracket\_line[bracket\_num]=nline;

bracket\_num++;

return bracket\_num;

}

int isMatching(char bracket[], int bracket\_num, int bracket\_line[])

{

int\* stack = (int\*)malloc(bracket\_num\*sizeof(int)); // 使用栈来存储左括号的索引

int top = -1; // 栈顶指针初始化为-1

// 遍历所有括号

for (int i = 0; i < bracket\_num; i++) {

// 如果是左括号，则将其索引压入栈中

if (bracket[i] == '(' || bracket[i] == '[' || bracket[i] == '{') {

top++;

stack[top] = i;

}

// 如果是右括号

else if (bracket[i] == ')' || bracket[i] == ']' || bracket[i] == '}') {

// 如果栈为空或栈顶的左括号与当前右括号不匹配，则返回false

if (top == -1 || (bracket[i] == ')' && bracket[stack[top]] != '(') ||

(bracket[i] == ']' && bracket[stack[top]] != '[') ||

(bracket[i] == '}' && bracket[stack[top]] != '{')) {

printf("Missing ");

switch (bracket[i]) {

case ')':

printf("left parenthesis '('.");

break;

case ']':

printf("left square bracket '['.");

break;

case '}':

printf("left curly brace '{'.");

break;

}

printf(" The other half is at line %d\n", bracket\_line[i]);

free(stack);

return 0;

}

// 匹配成功，将栈顶元素出栈

top--;

}

}

// 如果栈为空，则所有括号都匹配，否则返回false

if (top == -1) {

printf("All brackets are matching\n");

free(stack);

return 1;

} else {

printf("Missing ");

switch (bracket[stack[top]]) {

case '(':

printf("right parenthesis ')'");

break;

case '[':

printf("right square bracket ']'");

break;

case '{':

printf("right curly brace '}'");

break;

}

printf(" The other half is at line %d\n", bracket\_line[stack[top]]);

free(stack);

return 0;

}

}

%}

digit [0-9]

letter [a-zA-Z]

Ident ({letter}|\_)({letter}|{digit}|\_)\*

IntConst {digit}+

FloatConst {digit}+(\.{digit}+)?([eE][-+]?{digit}+)?

FormatString \"[^"]\*\"

MAIN [m][a][i][n]

CONST [c][o][n][s][t]

INT [i][n][t]

BREAK [b][r][e][a][k]

CONTINUE [c][o][n][t][i][n][u][e]

IF [i][f]

ELSE [e][l][s][e]

NOT !

AND &&

OR \|\|

WHILE [w][h][i][l][e]

GETINT [g][e][t][i][n][t]

PRINTF [p][r][i][n][t][f]

RETURN [r][e][t][u][r][n]

SWITCH [s][w][i][t][c][h]

CASE [c][a][s][e]

PLUS \+

MINU -

VOID [v][o][i][d]

MULT \\*

DIV [/]

MOD %

LSS <

LEQ <=

GRE >

GEQ >=

EQL ==

NEQ !=

ASSIGN =

COMMA ,

delim [ \t\r]

whitespace {delim}+

other .

%%

{MAIN} {printf("MAINTK %s\n",yytext);fprintf(output\_file, "MAINTK %s\n",yytext);}

{CONST} {printf("CONSTTK %s\n",yytext);fprintf(output\_file,"CONSTTK %s\n",yytext);}

{INT} {printf("INTTK %s\n",yytext);fprintf(output\_file, "INTTK %s\n",yytext);}

{BREAK} {printf("BREAKTK %s\n",yytext);fprintf(output\_file, "BREAKTK %s\n",yytext);}

{CONTINUE} {printf("CONTINUETK %s\n",yytext);fprintf(output\_file,"CONTINUETK %s\n",yytext);}

{IF} {printf("IFTK %s\n",yytext);fprintf(output\_file,"IFTK %s\n",yytext);}

{ELSE} {printf("ELSE %s\n",yytext);fprintf(output\_file,"ELSE %s\n",yytext);}

{WHILE} {printf("WHILETK %s\n",yytext);fprintf(output\_file,"WHILETK %s\n",yytext);}

{GETINT} {printf("GETINTTK %s\n",yytext);fprintf(output\_file,"GETINTTK %s\n",yytext);}

{PRINTF} {printf("PRINTFTK %s\n",yytext);fprintf(output\_file,"PRINTFTK %s\n",yytext);}

{RETURN} {printf("RETURNTK %s\n",yytext);fprintf(output\_file,"RETURNTK %s\n",yytext);}

{VOID} {printf("VOIDTK %s\n",yytext);fprintf(output\_file,"VOIDTK %s\n",yytext);}

{SWITCH} {printf("SWITCH %s\n",yytext);fprintf(output\_file,"SWITCH %s\n",yytext);}

{CASE} {printf("CASE %s\n",yytext);fprintf(output\_file,"CASE %s\n",yytext);}

{Ident} {printf("IDENFR %s\n",yytext);fprintf(output\_file,"IDENFR %s\n",yytext);}

{IntConst} {printf("INTCON %s\n",yytext);fprintf(output\_file,"INTCON %s\n",yytext);}

{FloatConst} {printf("FLOCON %s\n",yytext);fprintf(output\_file,"FLOCON %s\n",yytext);}

{AND} {printf("AND %s\n",yytext);fprintf(output\_file,"AND %s\n",yytext);}

{OR} {printf("OR %s\n",yytext);fprintf(output\_file,"OR %s\n",yytext);}

{LEQ} {printf("LEQ %s\n",yytext);fprintf(output\_file,"LEQ %s\n",yytext);}

{GEQ} {printf("GEQ %s\n",yytext);fprintf(output\_file,"GEQ %s\n",yytext);}

{EQL} {printf("EQL %s\n",yytext);fprintf(output\_file,"EQL %s\n",yytext);}

{NEQ} {printf("NEQ %s\n",yytext);fprintf(output\_file,"NEQ %s\n",yytext);}

{NOT} {printf("NOT %s\n",yytext);fprintf(output\_file,"NOT %s\n",yytext);}

{PLUS} {printf("PLUS %s\n",yytext);fprintf(output\_file,"PLUS %s\n",yytext);}

{MINU} {printf("MINU %s\n",yytext);fprintf(output\_file,"MINU %s\n",yytext);}

{MULT} {printf("MULT %s\n",yytext);fprintf(output\_file,"MULT %s\n",yytext);}

{DIV} {printf("DIV %s\n",yytext);fprintf(output\_file,"DIV %s\n",yytext);}

{MOD} {printf("MOD %s\n",yytext);fprintf(output\_file,"MOD %s\n",yytext);}

{LSS} {printf("LSS %s\n",yytext);fprintf(output\_file,"LSS %s\n",yytext);}

{GRE} {printf("GRE %s\n",yytext);fprintf(output\_file,"GRE %s\n",yytext);}

{ASSIGN} {printf("ASSIGN %s\n",yytext);fprintf(output\_file,"ASSIGN %s\n",yytext);}

{COMMA} {printf("COMMA %s\n",yytext);fprintf(output\_file,"COMMA %s\n",yytext);}

\n {nline++;}

{whitespace} { /\*不处理\*/ }

{FormatString} {printf("STRCON %s\n",yytext);fprintf(output\_file,"STRCON %s\n",yytext);}

"(" {

printf("LPARENT %s\n",yytext);fprintf(output\_file,"LPARENT %s\n",yytext);

bracket\_num = Set\_Brack(bracket, bracket\_line, bracket\_num, total\_len, yytext, nline);

}

")" {

printf("RPARENT %s\n",yytext);fprintf(output\_file,"RPARENT %s\n",yytext);

bracket\_num = Set\_Brack(bracket, bracket\_line, bracket\_num, total\_len, yytext, nline);

}

"[" {

printf("LBRACK %s\n",yytext);fprintf(output\_file,"LBRACK %s\n",yytext);

bracket\_num = Set\_Brack(bracket, bracket\_line, bracket\_num, total\_len, yytext, nline);

}

"]" {

printf("RBRACK %s\n",yytext);fprintf(output\_file,"RBRACK %s\n",yytext);

bracket\_num = Set\_Brack(bracket, bracket\_line, bracket\_num, total\_len, yytext, nline);

}

"{" {

printf("LBRACE %s\n",yytext);fprintf(output\_file,"LBRACE %s\n",yytext);

bracket\_num = Set\_Brack(bracket, bracket\_line, bracket\_num, total\_len, yytext, nline);

}

"}" {

printf("RBRACE %s\n",yytext);fprintf(output\_file,"RBRACE %s\n",yytext);

bracket\_num = Set\_Brack(bracket, bracket\_line, bracket\_num, total\_len, yytext, nline);

}

":" {printf("COLON %s\n",yytext);fprintf(output\_file,"COLON %s\n",yytext);}

";" {printf("SEMICN %s\n",yytext);fprintf(output\_file,"SEMICN %s\n",yytext);}

"#".\* {printf("HEAD %s\n",yytext);fprintf(output\_file,"HEAD %s\n",yytext);}

"//"(.)\*\n { nline++; }

"/\*"([^\*]|\\*[^/])\*"\\*/" {

// 统计其中的\n的个数

for(int i=0; i<strlen(yytext);i++){

if (yytext[i]=='\n')

{

nline++;

}

}

}

{other} {printf("illegal character, line %d:\'%s\'\n",nline,yytext);fprintf(output\_file,"illegal character, line %d:\'%s\'\n",nline,yytext);}

%%

int main(){

bracket = (char\*)malloc(sizeof(char)\*total\_len);

bracket\_line = (int\*)malloc(sizeof(int)\*total\_len);

yyin=fopen("test.c","r");

output\_file = fopen("output.txt", "w");

if (output\_file == NULL) {

printf("Error opening output file.\n");

return 1;

}

yylex();

isMatching(bracket, bracket\_num, bracket\_line);

printf("总行数为：%d",nline);

free(bracket);

free(bracket\_line);

return 0;

}

int yywrap(){

return 1;

}

#include <iostream>

#include <cstring>

#include <cstdio>

#include <algorithm>

#include <stack>

#include <fstream>

const int num\_of\_keyword=22;

const int num\_of\_MO=11;

const int num\_of\_BO=12;

const int num\_of\_D=9;

const char keyword[50][12]={"break","case","char","continue","do","default","double",

"else","float","for","if","int","include","long","main","return","switch","typedef","void","unsigned","while","iostream"};

const char Monocular\_Operator[20]={'+','-','\*','/','!','%','~','&','|','^','='}; //单目运算符

const char Binocular\_Operator[20][5]={"++","--","&&","||","<=","!=","==",">=","+=","-=","\*=","/="}; //双目运算符

const char Delimiter[20]={',','(',')','{','}',';','<','>','#'}; //界符

FILE\* file\_source=NULL;

bool IsInteger(char a)

{

if(a>='0' && a<='9')

return true;

return false;

}

bool IsLetter(char a)

{

if(a>='a' && a<='z')

return true;

if(a>='A' && a<='Z')

return true;

return false;

}

bool IsKeyword(char a[])

{

int len=strlen(a);

for(int j=0;j<num\_of\_keyword;++j)

{

if(strlen(keyword[j])==len)

{

if(strcmp(keyword[j],a)==0)

return true;

}

}

return false;

}

bool IsMO(char a)

{

for(int i=0;i<num\_of\_MO;++i)

{

if(Monocular\_Operator[i]==a)

return true;

}

return false;

}

bool IsBO(char a[])

{

for(int i=0;i<num\_of\_BO;++i)

{

if(strcmp(Binocular\_Operator[i],a)==0)

return true;

}

return false;

}

bool IsDelimiter(char a)

{

for(int i=0;i<num\_of\_D;++i)

{

if(Delimiter[i]==a)

return true;

}

return false;

}

void scan()

{

char str[10000];

char ch;

int i,j;

int point;

int flag;

ch=fgetc(file\_source);

bool finish=false;

while(!finish)

{

flag=-1;

point=0;

if(IsInteger(ch))

{

flag=1;

str[point++]=ch;

ch=fgetc(file\_source);

while(IsLetter(ch) || IsInteger(ch) || ch=='.' || ch=='+' || ch=='-')

{

flag=1;

str[point++]=ch;

ch=fgetc(file\_source);

}

str[point]='\0';

}

if(IsLetter(ch))

{

flag=2;

str[point++]=ch;

ch=fgetc(file\_source);

while(IsLetter(ch) || IsInteger(ch))

{

flag=2;

str[point++]=ch;

ch=fgetc(file\_source);

}

str[point]='\0';

}

if(flag==2)

{

if(IsKeyword(str))

{

std::cout<<str<<" "<<"关键字"<<std::endl;

}

point=0;

flag=-1;

}

if(IsDelimiter(ch))

{

std::cout<<ch<<" "<<"界符"<<std::endl;

if((ch=fgetc(file\_source))==EOF)

{

finish=true;

break;

}

}

if(IsMO(ch))

{

str[point++]=ch;

if((ch=fgetc(file\_source))==EOF)

{

finish=true;

}

str[point++]=ch;

str[point]='\0';

if(finish==false && IsBO(str))

{

std::cout<<str<<" "<<"双目运算符"<<std::endl;

ch=fgetc(file\_source);

}

else

{

std::cout<<str[0]<<" "<<"单目运算符"<<std::endl;

}

point=0;

}

if(ch==' ' || ch=='\n' || ch=='\t')

{

if((ch=fgetc(file\_source))==EOF)

{

finish=true;

break;

}

continue;

}

}

}

int main()

{

file\_source=fopen("test.txt","r+");

scan();

fclose(file\_source);

return 0;

}