实验三、词法分析实验

> *学号：1120180207*
>
> *姓名：唐小娟*
>
> *班级：07111801*

1. 实验目的

（1）熟悉 C 语言的词法规则，了解编译器词法分析器的主要功能和实现技术，掌握典型词法分析器构造方法，设计并实现 C 语言词法分析器；

（2）了解 Flex 工作原理和基本思想，学习使用工具自动生成词法分析器；

（3）掌握编译器从前端到后端各个模块的工作原理，词法分析模块与其他模块之间的交互过程。

2. 实验内容

根据 C 语言的词法规则，设计识别 C 语言所有单词类的词法分析器的确定有限状态自动机，并使用 Java、C\C++或者 Python 其中任何一种语言，采用程序中心法或者数据中心法设计并实现词法分析器。

词法分析器的输入为 C 语言源程 序，输出为属性字流。

学生可以选择编码实现词法分析器，也可以选择使用 Flex 自动生成词法分析器。需要注意的是，Flex 生成的是 C 为实现语言的词法分析器，如果需要生成 Java 为实现语言的词法分析器，可以尝试 JFlex 或者 ANTLR。 由于框架是基于 Java 语言实现的，并且提供了相应的示例程序，建议学生 使用 Java 语言在示例的基础上完成词法分析器。

本实验我采用Flex来生成以C为实现语言的词法生成器，其生成的代码文件为lex.yy.c，之后用gcc进行编译得到词法生生成器的可执行文件myscanner.exe。

3. 实验环境

名称	信息
操作系统版本	Ubuntu 20.04.1 LTS、Windows家庭中文版
Flex版本	2.6.4
GCC版本	8.1.0
gedit版本	3.36.2

4. 实验过程

该实验以 C 语言作为源语言，构建 C 语言的词法分析器，对于给定的测试程 序，输出属性字符流。词法分析器的构建按照 C 语言的词法规则进行。C 语言的 发展经历了不同的阶段，早期按照 C99 标准进行编程和编译器的实现，2011 年 又对 C 语言规范进行了修订，形成了 C11（又称 C1X）。下面以 C11 为基准，对 C 语言的词法规则进行简要的描述和相关设计。

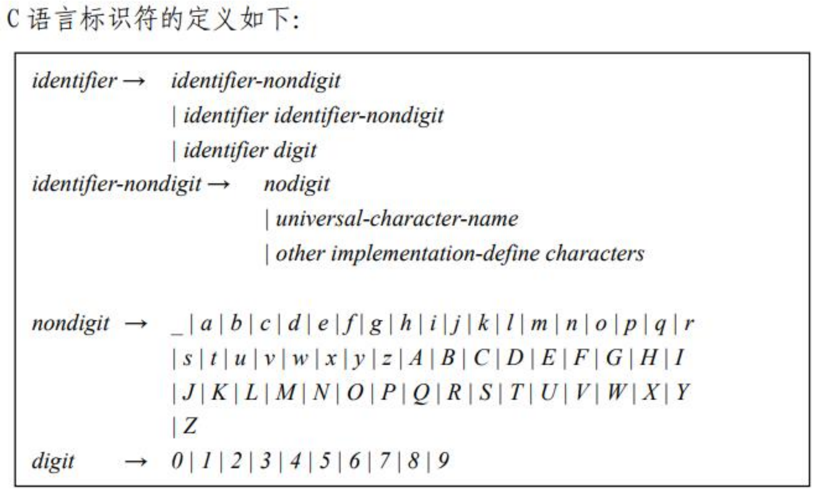
4.1 设计思路

- 对语言的各类单词分别构造状态图；
- 将各类状态图进行合并，构成一个能识别该语言所有单词的状态图；
- 将各类单词的状态图的初态合并为唯一初态；
- 调整冲突的编号

  在这里，我分成了三类单词：标识符、整数常量、浮点数常量、字符常量、字符串常量、运算符和界限符。为了简化实现过程，具体的定义以老师给定文档为标准。

4.1.1标识符

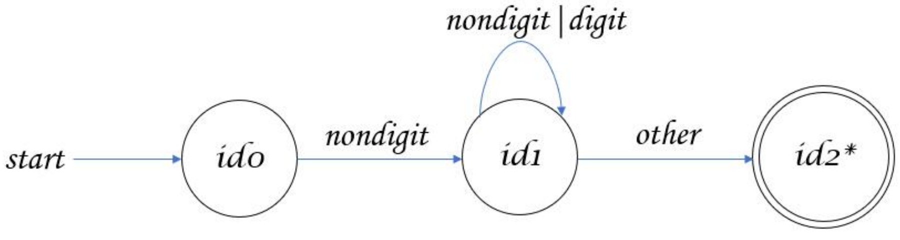
定义：只能由字母、数字和下划线三种字符组成，且第一个字符必须是字母或下划线。



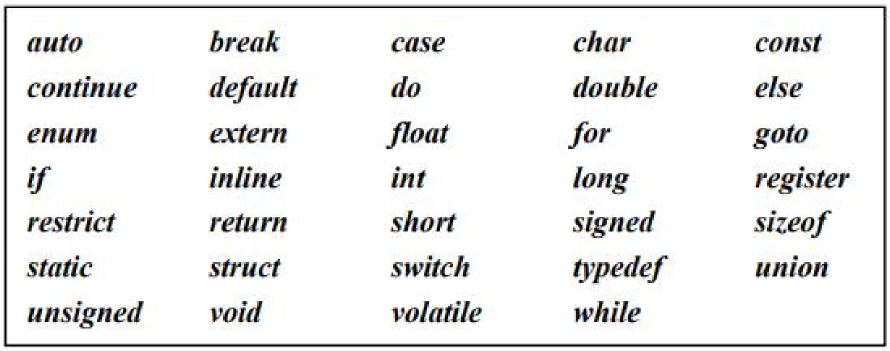
C 语言标识符的定义如下:

```
identifier →    identifier-nondigit
                | identifier identifier-nondigit
                | identifier digit
identifier-nondigit →   nodigit
                        | universal-character-name
                        | other implementation-define characters

nondigit  →    _ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r
               | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I
               | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y
               | Z
digit     →    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

DFA：



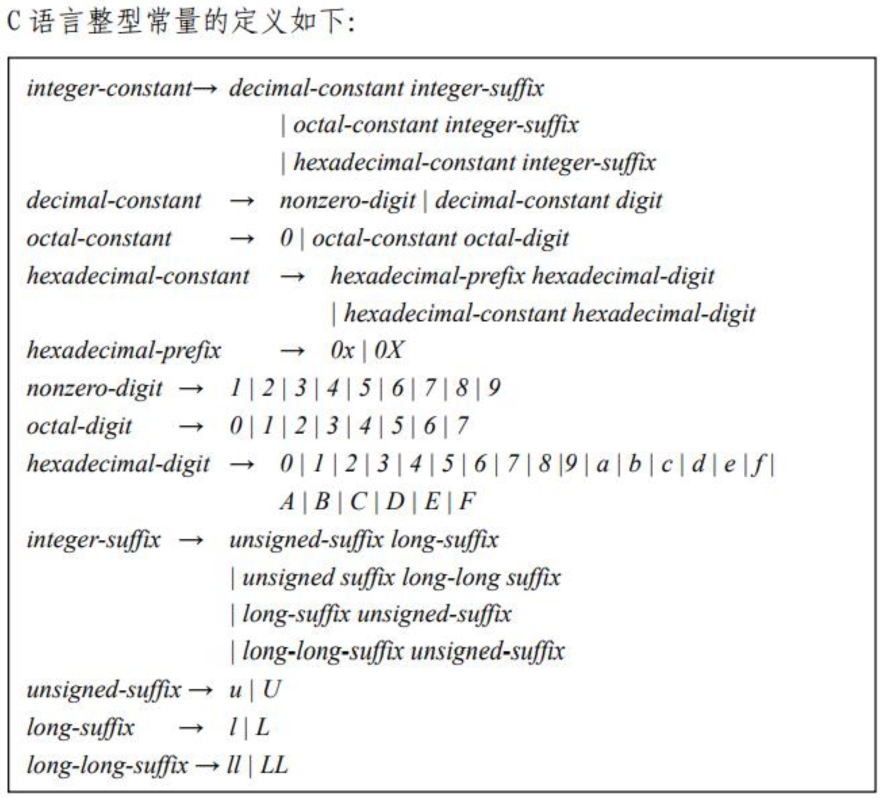
其中，other代表非nondigit|digit的字符，状态2为结束状态，且代表回退一个字符。除此之外，由于关键字也满足上述DFA，所以还需要进行关键字的筛选。



auto	***break***	***case***	***char***	***const***
continue	***default***	***do***	***double***	***else***
enum	***extern***	***float***	***for***	***goto***
if	***inline***	***int***	***long***	***register***
restrict	***return***	***short***	***signed***	***sizeof***
static	***struct***	***switch***	***typedef***	***union***
unsigned	***void***	***volatile***	***while***	

4.1.2 整形常量

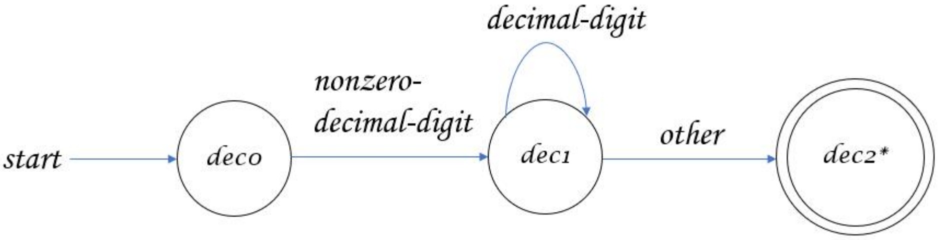
定义：C语言的整形常量主要包括：十进制、八进制、十六进制。依照这三类的定义分别实现DFA。



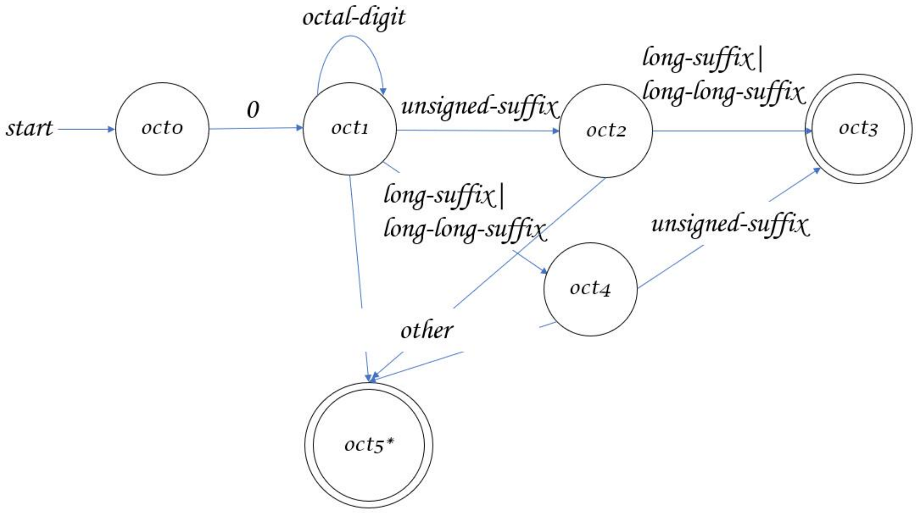
C 语言整型常量的定义如下:

integer-constant → *decimal-constant integer-suffix*
                | *octal-constant integer-suffix*
                | *hexadecimal-constant integer-suffix*
decimal-constant → *nonzero-digit* | *decimal-constant digit*
octal-constant → *0* | *octal-constant octal-digit*
hexadecimal-constant → *hexadecimal-prefix hexadecimal-digit*
                | *hexadecimal-constant hexadecimal-digit*
hexadecimal-prefix → *0x* | *0X*
nonzero-digit → *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*
octal-digit → *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7*
hexadecimal-digit → *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |*9* | *a* | *b* | *c* | *d* | *e* | *f* |
                A | *B* | *C* | *D* | *E* | *F*
integer-suffix → *unsigned-suffix long-suffix*
                | *unsigned suffix long-long suffix*
                | *long-suffix unsigned-suffix*
                | *long-long-suffix unsigned-suffix*
unsigned-suffix → *u* | *U*
long-suffix → *l* | *L*
long-long-suffix → *ll* | *LL*

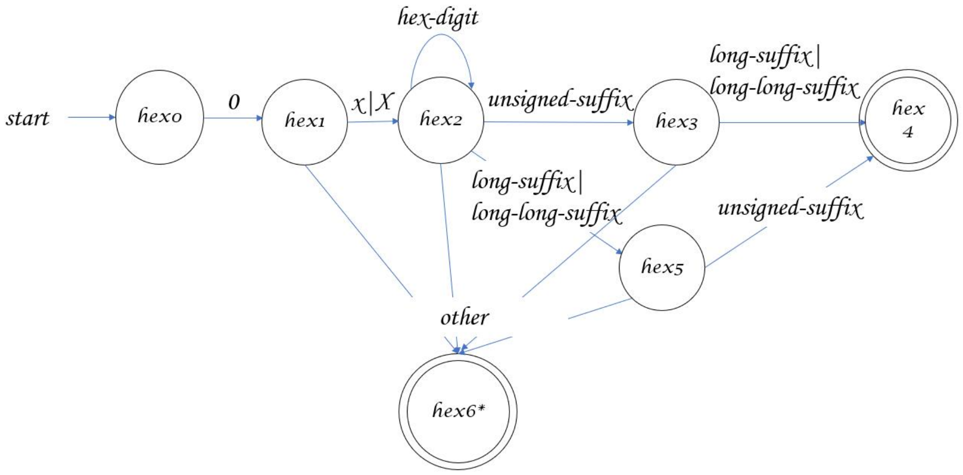
decimal-constant



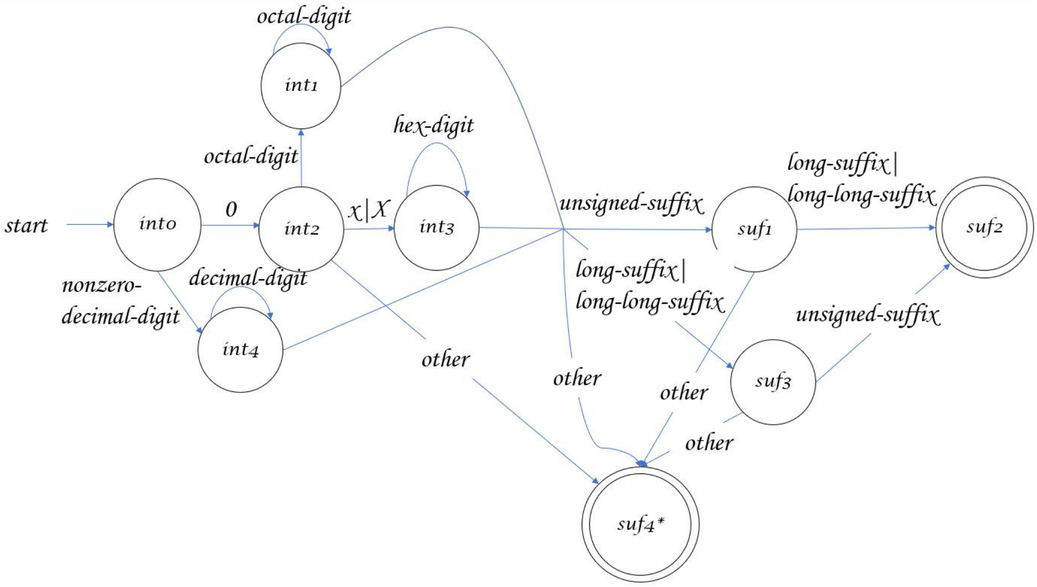
octal-constant



hexadecimal-constant

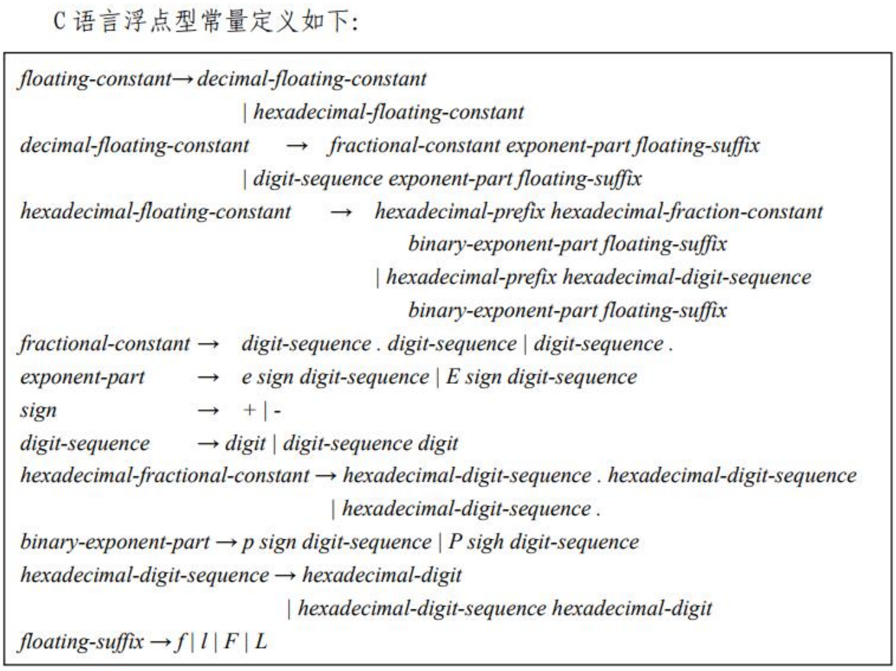


所以，将这三个DFA结合起来，有Interger的DFA：



4.1.3 浮点数常量

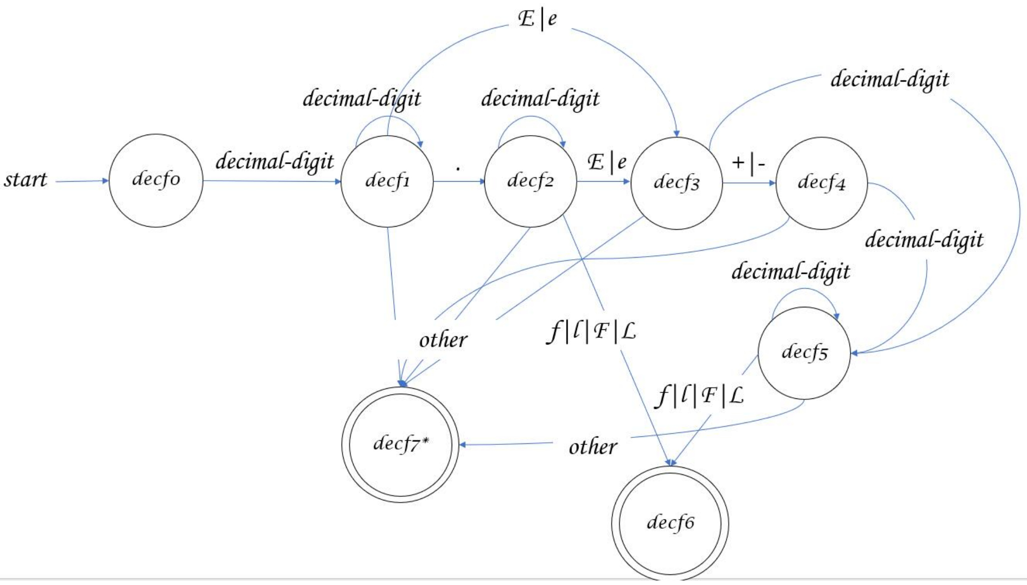
定义：C语言的浮点数常量主要包括：十进制、十六进制。依照这三类的定义分别实现DFA。



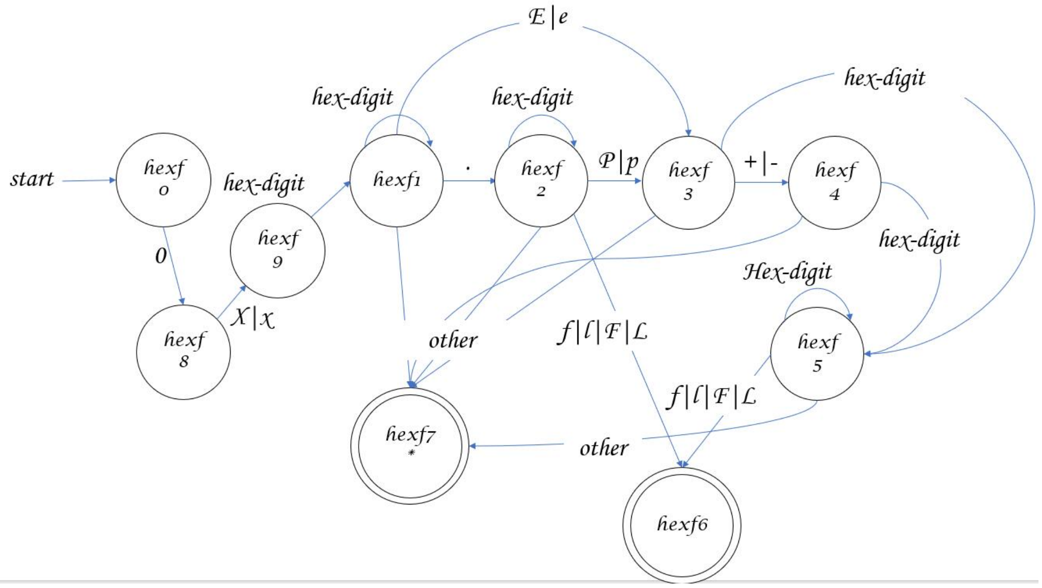
C 语言浮点型常量定义如下:

```
floating-constant→ decimal-floating-constant
                        | hexadecimal-floating-constant
decimal-floating-constant    →  fractional-constant exponent-part floating-suffix
                        | digit-sequence exponent-part floating-suffix
hexadecimal-floating-constant    →  hexadecimal-prefix hexadecimal-fraction-constant
                                        binary-exponent-part floating-suffix
                                     | hexadecimal-prefix hexadecimal-digit-sequence
                                        binary-exponent-part floating-suffix
fractional-constant →   digit-sequence . digit-sequence | digit-sequence .
exponent-part       →   e sign digit-sequence | E sign digit-sequence
sign                →   + | -
digit-sequence      → digit | digit-sequence digit
hexadecimal-fractional-constant → hexadecimal-digit-sequence . hexadecimal-digit-sequence
                              | hexadecimal-digit-sequence .
binary-exponent-part → p sign digit-sequence | P sigh digit-sequence
hexadecimal-digit-sequence → hexadecimal-digit
                           | hexadecimal-digit-sequence hexadecimal-digit
floating-suffix → f | l | F | L
```

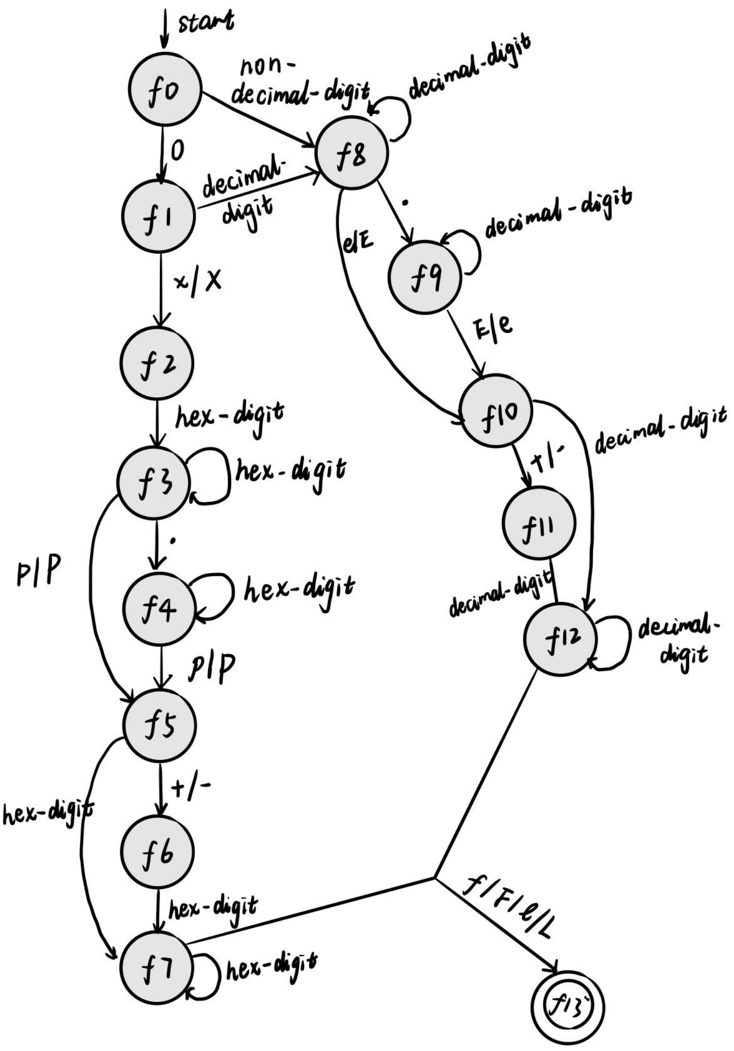
decimal-floating-constant



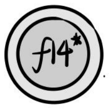
hexadecimal-floating-constant



所以将这两个DFA结合起来，得到float的DFA：

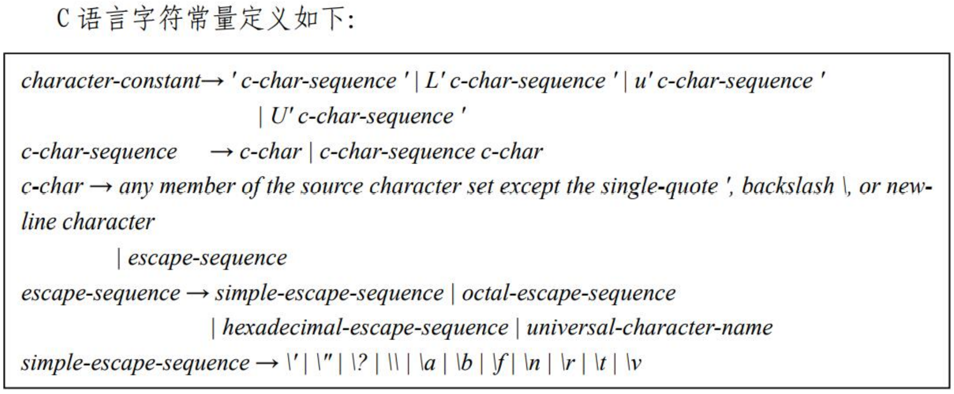


除f0、f13*外，其他状态均有一条other线连入



4.1.4 字符常量

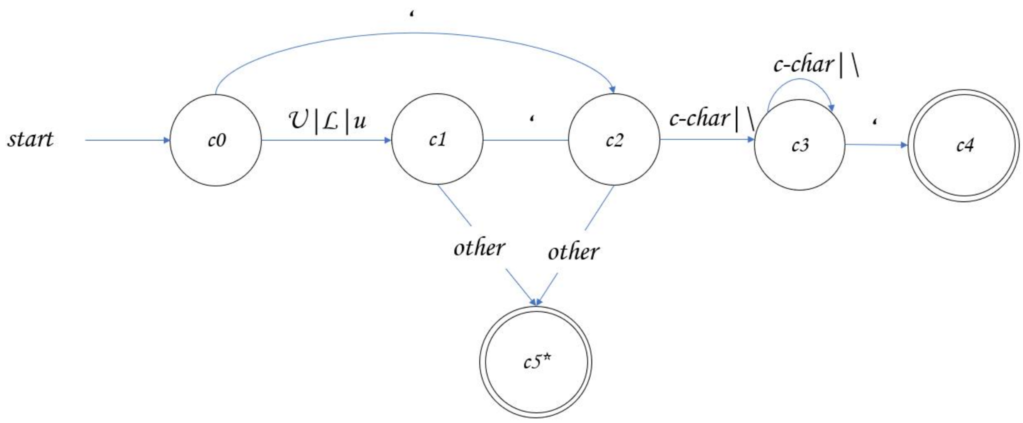
经过查阅资料，可知字符可以有多个字符，如'abcd'，虽然会有warning，但编译执行仍然是通过的。在这里为了简便，就不考虑出错处理了。简单的认为单引号引起的内容就是字符常量。



C 语言字符常量定义如下:

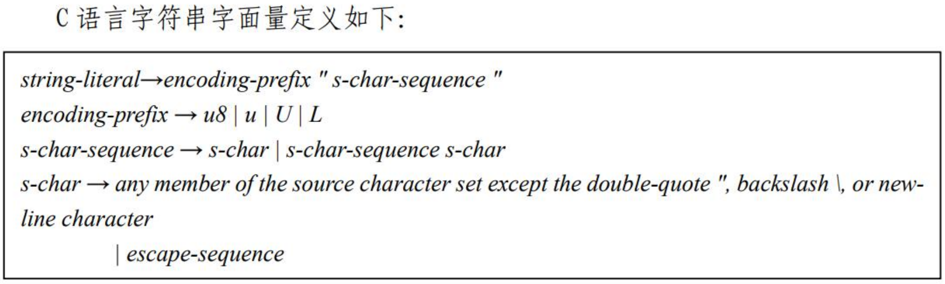
character-constant→ ' c-char-sequence ' | L' c-char-sequence ' | u' c-char-sequence '
| U' c-char-sequence '
c-char-sequence → c-char | c-char-sequence c-char
c-char → any member of the source character set except the single-quote ', backslash \, or new-line character
| escape-sequence
escape-sequence → simple-escape-sequence | octal-escape-sequence
| hexadecimal-escape-sequence | universal-character-name
simple-escape-sequence → \' | \" | \? | \\ | \a | \b | \f | \n | \r | \t | \v

DFA为：



4.1.5 字符串常量

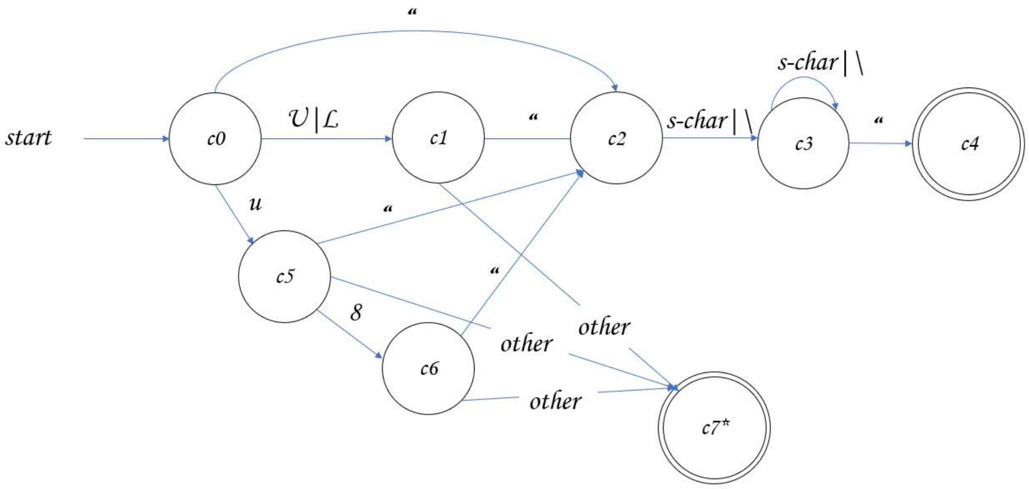
定义如下：



C 语言字符串字面量定义如下:

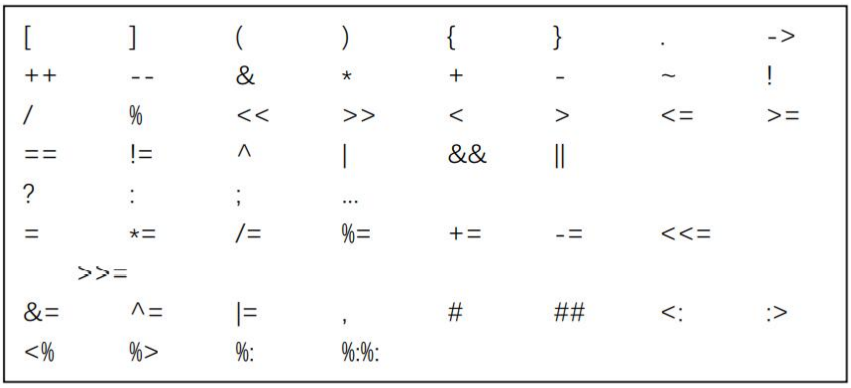
string-literal→encoding-prefix " s-char-sequence "
encoding-prefix → u8 | u | U | L
s-char-sequence → s-char | s-char-sequence s-char
s-char → any member of the source character set except the double-quote ", backslash \, or new-line character
| escape-sequence

DFA：



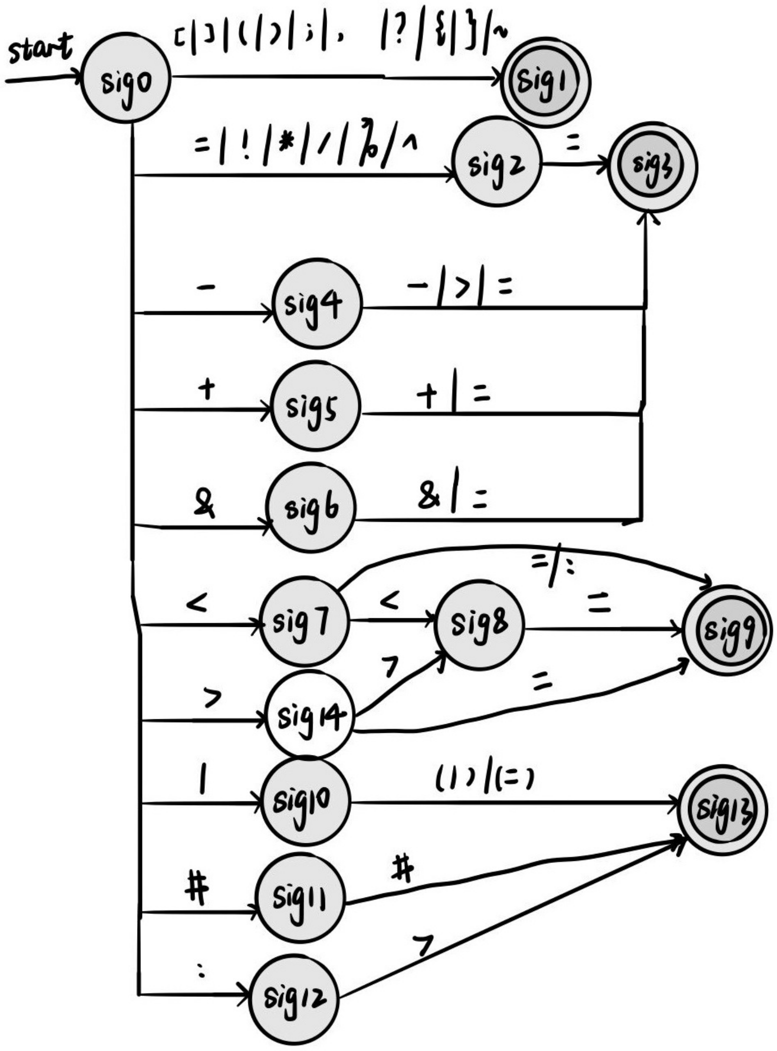
4.1.6 运算符和界限符

根据文档定义的运算符和界限符内容：由于<%、%>、%:、%:%:、…并不常用，所以就忽略了。



[	]	(	)	{	}	.	->
++	--	&	*	+	-	~	!
/	%	<<	>>	<	>	<=	>=
==	!=	^	|	&&	||		
?	:	;	...				
=	*=	/=	%=	+=	-=	<<=	
>>=							
&=	^=	|=	,	#	##	<:	:>
<%	%>	%:	%:%:				

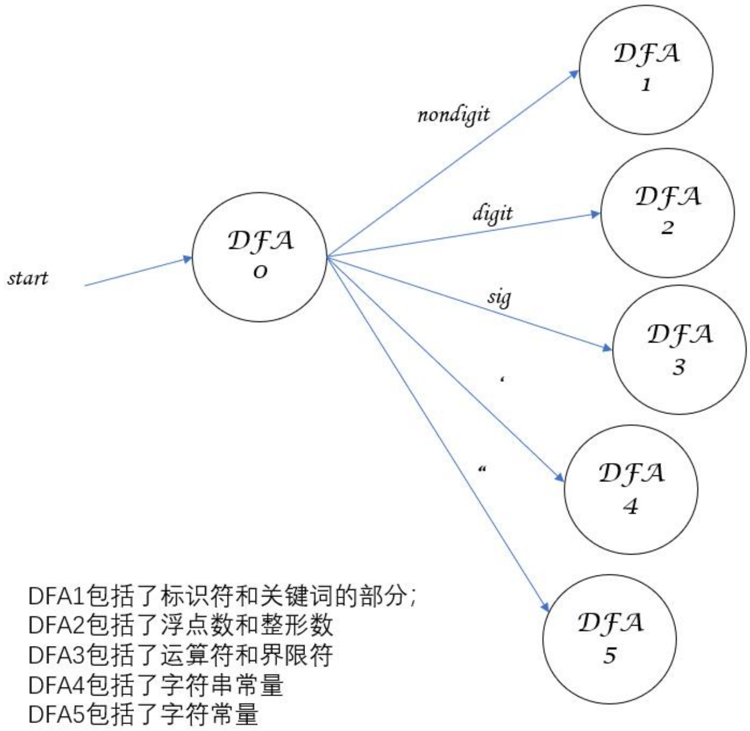
DFA如下：



除sig0、sig1、sig3、sig9、sig13结束状态外，其他所有状态均有一条other线连到 sig15* 状态.

4.1.7 总DFA

将前文所有设计的 DFA 合并到一起称为最终的 DFA，由于汇总起来得到的DFA实在过于庞大，因此将这个整个大的DFA先分成五个子图。如下：（具体的不再示意）



DFA1包括了标识符和关键词的部分;
DFA2包括了浮点数和整形数
DFA3包括了运算符和界限符
DFA4包括了字符串常量
DFA5包括了字符常量

4.2 代码实现

本次实验采取Flex自动生成词法分析器，关键部分在于上述单词正则表达式的书写，除此之外，在匹配的过程中，从上到下优先级从高到低，也就是说，先匹配到上方的正则表达式之后，就不会再进行匹配了。例如：匹配关键词要先于标识符。myscanner.l具体部分关键代码如下：

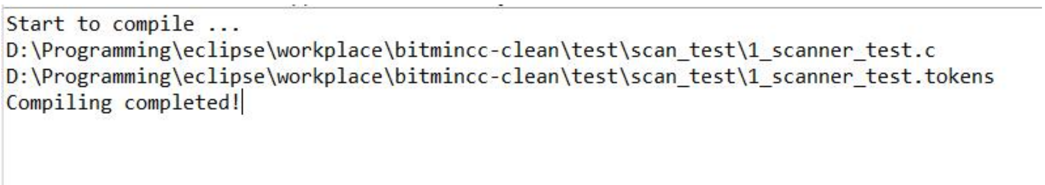
```
KEYWORD (auto)|(break)|(case)|(char)|(const)|(continue)|(default)|(do)|(double)|(else)|(enum)|(extern)|(float)|(for)|(goto)|(if)|(inline)|(int)|(long)|(register)|(restrict)|(return)|(short)|(signed)|(sizeof)|(static)|(struct)|(switch)|(typedef)|(union)|(unsigned)|(void)|(volatile)|(while)

IDENTIFIER ([_a-zA-Z][_a-zA-Z0-9]*)

THREESYMBOL [<][<][=]|[>][>][=]

DOUBLESIMBOL [\*\+\^\|=<>!&%-][=]|[-][>]|[\+][\+]|[-][-]|[<][<]|[>][>]|[&][&]|[\|][\|]|[#][#]

SIMBOL  ([\[\](),.;:!#{}\-?^<>&~|%*+/=])

CHAR    ([cLuU]?)(\'.*\')

STRING  (([uUL]?)|(u8))(\"(.*)\")

INTERGER (([1-9][0-9]*)|(0[0-7]*)|(0[xX][0-9a-fA-F]*))([uU]?(ll|LL)?|(ll|LL)?[uU]?|[uU]?[Ll]?|[Ll]?[Uu]?)

DECIFLOAT ([0-9]+\.?[0-9]*)([eE][+-]?[0-9]+)?([flFL])?
```

```
HEXAFLOAT (0[xX][0-9a-fA-F]+\.?[0-9a-fA-F]*)([pP][+-]?[0-9]+)?([flFL])?
COMMENT   (\/\/.*)

{KEYWORD}       {

                    numStart = temp+1;
                    numEnd=numStart + strlen(yytext)-1;
                    temp = numEnd;
                    numTokens++;
                    return T_keyword;
                }

//...


int main(int argc,char* argv[])
{
    int token_type;

    yyin=fopen(argv[1],"r");
    yyout=fopen(argv[2],"w");
    printf("%s\n",argv[1]);
    printf("%s\n",argv[2]);
    while (token_type = yylex()) {
        if(token_type == T_empty) continue;
        if(token_type == T_symbol)
            fprintf(yyout,"[@%d,%d:%d=\'%s\',<'%s'>,%d:%d]\n",numTokens,numStart,numEnd,yytext,yytext,numLines,numStart);
        else
            fprintf(yyout,"[@%d,%d:%d=\'%s\',<%s>,%d:%d]\n",numTokens,numStart,numEnd,yytext,token_strs[token_type],numLines,numStart);
    }
    return 0;
}
```

myscanner.h的接口文件（方便输出相关内容）：

```
#ifndef MYSCANNER_H
#define MYSCANNER_H




typedef enum{
T_over,T_keyword,T_identifier,T_interger,T_float,T_char,T_string,T_symbol,T_empty

}TokenType;

const char* token_strs[]={
"EOF","keyword","Identifier","const_interger","const_float","const_char","const_string"
};


#endif

```

之后运行flex myscanner.l生成的lex.yy.c源文件，再通过gcc编译生成可执行文件，修改老师给定的BIT-MiniCC框架中的config.xml文件（type和path），运行之后，可以得到相应的tokens文件。



```
Start to compile ...
D:\Programming\eclipse\workplace\bitmincc-clean\test\scan_test\1_scanner_test.c
D:\Programming\eclipse\workplace\bitmincc-clean\test\scan_test\1_scanner_test.tokens
Compiling completed!
```

5. 实验结果

根据老师给定的测试文件1_scanner_test.c文件

```
int main()
{
    //integer-constant
    +1 ;
    -10 ;
    1000 ;
    1l ;
    1U ;
    10ul ;
    10LU ;
    1000ull ;
    1000LLU ;

    +0 ;
    -00 ;
    007 ;
    00ul ;
    00LLU ;

    +0x0 ;
    -0x00 ;
    0XABCDEF ;
    0xful ;
    0XFLLU ;

```

```
//floating-constant
    0.0 ;
    +1.1e+1 ;
    -1.1E-1 ;
    1.1e1f ;
    1.1E1L ;

    0x0p0;
    0x0.0p0 ;
    +0xa.ap+1 ;
    -0xa.aP-1 ;
    0xa.ap1f ;
    0xa.aP1L ;

//character-constant
    'a' ;
    L'a' ;
    U'a' ;
    u'a' ;
    '\n' ;
    '\?' ;
    '\24' ;

//string-literal
    "abcdefg123456\\" ;
    u8"a" ;
    u"a" ;
    U"a" ;
    L"a" ;

//identifier
    int __a ;
    int __1 ;
    int a1 ;
    int a ;
    int a_1 ;
    int a_ ;

//keyword
    int i=1 ;
    float f ;
    double d ;
    char c ;
    long l ;
    short s ;
    signed si ;
    unsigned short us ;
    typedef struct
    {
        int num1;
    }test;
    test test1 ;
    test *test2 ;
    static int sti ;
    const int ci ;
    sizeof( int ) ;
    if( 1 )
    {

    }
    else
    {

    }
    for( ; ; )
    {
        continue ;
    }
    while( 1 )
    {

    }
    switch(i){
        case 1:  break;
        default:  break;
    }
    do
    {

    }while( 0 ) ;
    goto gotoFlag ;
    //operators
    int array[10] = {0} ;
    test1.num1 = 0 ;
    test2->num1 = 0 ;
    i++ ;
    i-- ;
    i = 1 + 1 ;
    i = 1 - 1 ;
    i = 1 * 1 ;
    i = 1 / 1 ;
    i = 1 % 1 ;
    i = !i ;
    i = i & 1 ;
    i = i | 1 ;
    i = i && 1 ;
    i = i || 1 ;
    i = i ^ 1 ;
    i = i >> 1 ;
    i = i << 1 ;
    i = i > 1 ? 1 : 2 ;
    i += 1 ;
    i -= 1 ;
    i *= 1 ;
    i /= 1 ;
    i %= 1 ;
    i &= 1 ;
    i |= 1 ;
    i ^= 1 ;
    i >>= 1 ;
    i <<= 1 ;
    if( i==1 ){}
    if( i!=1 ){}
    if( i>1 ){}
    if( i<1 ){}
    if( i>=1 ){}
    if( i<=1 ){}
gotoFlag:
    return 0 ;
}
void function( int arg1 , int arg2 )
{
}
```

得到我们的词法分析结果：

```
[@0,0:2='int',<keyword>,1:0]
[@1,4:7='main',<Identifier>,1:4]
[@2,8:8='(',<'('>,1:8]
[@3,9:9=')',<')'>,1:9]
[@4,0:0='{',<'{'>,2:0]
[@5,1:1='+',<'+'>,4:1]
[@6,2:2='1',<const_interger>,4:2]
[@7,4:4=';',<';'>,4:4]
[@8,1:1='-',<'-'>,5:1]
[@9,2:3='10',<const_interger>,5:2]
[@10,5:5=';',<';'>,5:5]
[@11,1:4='1000',<const_interger>,6:1]
[@12,6:6=';',<';'>,6:6]
[@13,1:2='1l',<const_interger>,7:1]
[@14,4:4=';',<';'>,7:4]
[@15,1:2='1U',<const_interger>,8:1]
[@16,4:4=';',<';'>,8:4]
[@17,1:4='10ul',<const_interger>,9:1]
[@18,6:6=';',<';'>,9:6]
[@19,1:4='10LU',<const_interger>,10:1]
[@20,6:6=';',<';'>,10:6]
[@21,1:7='1000ull',<const_interger>,11:1]
[@22,9:9=';',<';'>,11:9]
[@23,1:7='1000LLU',<const_interger>,12:1]
[@24,9:9=';',<';'>,12:9]
[@25,1:1='+',<'+'>,14:1]
[@26,2:2='0',<const_interger>,14:2]
[@27,4:4=';',<';'>,14:4]
[@28,1:1='-',<'-'>,15:1]
```

```
[@29,2:3='00',<const_interger>,15:2]
[@30,5:5=';',<';'>,15:5]
[@31,1:3='007',<const_interger>,16:1]
[@32,5:5=';',<';'>,16:5]
[@33,1:4='00ul',<const_interger>,17:1]
[@34,6:6=';',<';'>,17:6]
[@35,1:5='00LLU',<const_interger>,18:1]
[@36,7:7=';',<';'>,18:7]
[@37,1:1='+',<'+'>,20:1]
[@38,2:4='0x0',<const_interger>,20:2]
[@39,6:6=';',<';'>,20:6]
[@40,1:1='-',<'-'>,21:1]
[@41,2:5='0x00',<const_interger>,21:2]
[@42,7:7=';',<';'>,21:7]
[@43,1:8='0XABCDEF',<const_interger>,22:1]
[@44,10:10=';',<';'>,22:10]
[@45,1:5='0xful',<const_interger>,23:1]
[@46,7:7=';',<';'>,23:7]
[@47,1:6='0XFLLU',<const_interger>,24:1]
[@48,8:8=';',<';'>,24:8]
[@49,1:3='0.0',<const_float>,27:1]
[@50,5:5=';',<';'>,27:5]
[@51,1:1='+',<'+'>,28:1]
[@52,2:7='1.1e+1',<const_float>,28:2]
[@53,9:9=';',<';'>,28:9]
[@54,1:1='-',<'-'>,29:1]
[@55,2:7='1.1E-1',<const_float>,29:2]
[@56,9:9=';',<';'>,29:9]
[@57,1:6='1.1e1f',<const_float>,30:1]
[@58,8:8=';',<';'>,30:8]
[@59,1:6='1.1E1L',<const_float>,31:1]
[@60,8:8=';',<';'>,31:8]
[@61,1:5='0x0p0',<const_float>,33:1]
[@62,6:6=';',<';'>,33:6]
[@63,1:7='0x0.0p0',<const_float>,34:1]
[@64,9:9=';',<';'>,34:9]
[@65,1:1='+',<'+'>,35:1]
[@66,2:9='0xa.ap+1',<const_float>,35:2]
[@67,11:11=';',<';'>,35:11]
[@68,1:1='-',<'-'>,36:1]
[@69,2:9='0xa.aP-1',<const_float>,36:2]
[@70,11:11=';',<';'>,36:11]
[@71,1:8='0xa.ap1f',<const_float>,37:1]
[@72,10:10=';',<';'>,37:10]
[@73,1:8='0xa.aP1L',<const_float>,38:1]
[@74,10:10=';',<';'>,38:10]
[@75,1:3=''a'',<const_char>,41:1]
[@76,5:5=';',<';'>,41:5]
[@77,1:4='L'a'',<const_char>,42:1]
[@78,6:6=';',<';'>,42:6]
[@79,1:4='U'a'',<const_char>,43:1]
[@80,6:6=';',<';'>,43:6]
```

```
[@81,1:4='u'a'',<const_char>,44:1]
[@82,6:6=';',<';'>,44:6]
[@83,1:4=''\n'',<const_char>,45:1]
[@84,6:6=';',<';'>,45:6]
[@85,1:4=''\?'',<const_char>,46:1]
[@86,6:6=';',<';'>,46:6]
[@87,1:5=''\24'',<const_char>,47:1]
[@88,7:7=';',<';'>,47:7]
[@89,1:17='"abcdefg123456\\"',<const_string>,50:1]
[@90,19:19=';',<';'>,50:19]
[@91,1:5='u8"a"',<const_string>,51:1]
[@92,7:7=';',<';'>,51:7]
[@93,1:4='u"a"',<const_string>,52:1]
[@94,6:6=';',<';'>,52:6]
[@95,1:4='U"a"',<const_string>,53:1]
[@96,6:6=';',<';'>,53:6]
[@97,1:4='L"a"',<const_string>,54:1]
[@98,6:6=';',<';'>,54:6]
[@99,1:3='int',<keyword>,57:1]
[@100,5:7='__a',<Identifier>,57:5]
[@101,9:9=';',<';'>,57:9]
[@102,1:3='int',<keyword>,58:1]
[@103,5:7='__1',<Identifier>,58:5]
[@104,9:9=';',<';'>,58:9]
[@105,1:3='int',<keyword>,59:1]
[@106,5:6='a1',<Identifier>,59:5]
[@107,8:8=';',<';'>,59:8]
[@108,1:3='int',<keyword>,60:1]
[@109,5:5='a',<Identifier>,60:5]
[@110,7:7=';',<';'>,60:7]
[@111,1:3='int',<keyword>,61:1]
[@112,5:7='a_1',<Identifier>,61:5]
[@113,9:9=';',<';'>,61:9]
[@114,1:3='int',<keyword>,62:1]
[@115,5:6='a_',<Identifier>,62:5]
[@116,8:8=';',<';'>,62:8]
[@117,1:3='int',<keyword>,65:1]
[@118,5:5='i',<Identifier>,65:5]
[@119,6:6='=',<'='>,65:6]
[@120,7:7='1',<const_interger>,65:7]
[@121,9:9=';',<';'>,65:9]
[@122,1:5='float',<keyword>,66:1]
[@123,7:7='f',<Identifier>,66:7]
[@124,9:9=';',<';'>,66:9]
[@125,1:6='double',<keyword>,67:1]
[@126,8:8='d',<Identifier>,67:8]
[@127,10:10=';',<';'>,67:10]
[@128,1:4='char',<keyword>,68:1]
[@129,6:6='c',<Identifier>,68:6]
[@130,8:8=';',<';'>,68:8]
[@131,1:4='long',<keyword>,69:1]
[@132,6:6='l',<Identifier>,69:6]
```

```
[@133,8:8=';',<';'>,69:8]
[@134,1:5='short',<keyword>,70:1]
[@135,7:7='s',<Identifier>,70:7]
[@136,9:9=';',<';'>,70:9]
[@137,1:6='signed',<keyword>,71:1]
[@138,8:9='si',<Identifier>,71:8]
[@139,11:11=';',<';'>,71:11]
[@140,1:8='unsigned',<keyword>,72:1]
[@141,10:14='short',<keyword>,72:10]
[@142,16:17='us',<Identifier>,72:16]
[@143,19:19=';',<';'>,72:19]
[@144,1:7='typedef',<keyword>,73:1]
[@145,9:14='struct',<keyword>,73:9]
[@146,1:1='{',<'{'>,74:1]
[@147,2:4='int',<keyword>,75:2]
[@148,6:9='num1',<Identifier>,75:6]
[@149,10:10=';',<';'>,75:10]
[@150,1:1='}',<'}'>,76:1]
[@151,2:5='test',<Identifier>,76:2]
[@152,6:6=';',<';'>,76:6]
[@153,1:4='test',<Identifier>,77:1]
[@154,6:10='test1',<Identifier>,77:6]
[@155,12:12=';',<';'>,77:12]
[@156,1:4='test',<Identifier>,78:1]
[@157,6:6='*',<'*'>,78:6]
[@158,7:11='test2',<Identifier>,78:7]
[@159,13:13=';',<';'>,78:13]
[@160,1:6='static',<keyword>,79:1]
[@161,8:10='int',<keyword>,79:8]
[@162,12:14='sti',<Identifier>,79:12]
[@163,16:16=';',<';'>,79:16]
[@164,1:5='const',<keyword>,80:1]
[@165,7:9='int',<keyword>,80:7]
[@166,11:12='ci',<Identifier>,80:11]
[@167,14:14=';',<';'>,80:14]
[@168,1:6='sizeof',<keyword>,81:1]
[@169,7:7='(',<'('>,81:7]
[@170,9:11='int',<keyword>,81:9]
[@171,13:13=')',<')'>,81:13]
[@172,15:15=';',<';'>,81:15]
[@173,1:2='if',<keyword>,82:1]
[@174,3:3='(',<'('>,82:3]
[@175,5:5='1',<const_interger>,82:5]
[@176,7:7=')',<')'>,82:7]
[@177,1:1='{',<'{'>,83:1]
[@178,1:1='}',<'}'>,85:1]
[@179,1:4='else',<keyword>,86:1]
[@180,1:1='{',<'{'>,87:1]
[@181,1:1='}',<'}'>,89:1]
[@182,1:3='for',<keyword>,90:1]
[@183,4:4='(',<'('>,90:4]
[@184,6:6=';',<';'>,90:6]
[@185,8:8=';',<';'>,90:8]
[@186,10:10=')',<')'>,90:10]
[@187,1:1='{',<'{'>,91:1]
[@188,2:9='continue',<keyword>,92:2]
[@189,11:11=';',<';'>,92:11]
[@190,1:1='}',<'}'>,93:1]
[@191,1:5='while',<keyword>,94:1]
[@192,6:6='(',<'('>,94:6]
[@193,8:8='1',<const_interger>,94:8]
[@194,10:10=')',<')'>,94:10]
[@195,1:1='{',<'{'>,95:1]
[@196,1:1='}',<'}'>,97:1]
[@197,1:6='switch',<keyword>,98:1]
[@198,7:7='(',<'('>,98:7]
[@199,8:8='i',<Identifier>,98:8]
[@200,9:9=')',<')'>,98:9]
[@201,10:10='{',<'{'>,98:10]
[@202,5:8='case',<keyword>,99:5]
[@203,10:10='1',<const_interger>,99:10]
[@204,11:11=':',<':'>,99:11]
[@205,14:18='break',<keyword>,99:14]
[@206,19:19=';',<';'>,99:19]
[@207,5:11='default',<keyword>,100:5]
[@208,12:12=':',<':'>,100:12]
[@209,15:19='break',<keyword>,100:15]
[@210,20:20=';',<';'>,100:20]
[@211,1:1='}',<'}'>,101:1]
[@212,1:2='do',<keyword>,102:1]
[@213,1:1='{',<'{'>,103:1]
[@214,1:1='}',<'}'>,105:1]
[@215,2:6='while',<keyword>,105:2]
[@216,7:7='(',<'('>,105:7]
[@217,9:9='0',<const_interger>,105:9]
[@218,11:11=')',<')'>,105:11]
[@219,13:13=';',<';'>,105:13]
[@220,1:4='goto',<keyword>,106:1]
[@221,6:13='gotoFlag',<Identifier>,106:6]
[@222,15:15=';',<';'>,106:15]
[@223,1:3='int',<keyword>,108:1]
[@224,5:9='array',<Identifier>,108:5]
[@225,10:10='[',<'['>,108:10]
[@226,11:12='10',<const_interger>,108:11]
[@227,13:13=']',<']'>,108:13]
[@228,15:15='=',<'='>,108:15]
[@229,17:17='{',<'{'>,108:17]
[@230,18:18='0',<const_interger>,108:18]
[@231,19:19='}',<'}'>,108:19]
[@232,21:21=';',<';'>,108:21]
[@233,1:5='test1',<Identifier>,109:1]
[@234,6:6='.',<'.'>,109:6]
[@235,7:10='num1',<Identifier>,109:7]
[@236,12:12='=',<'='>,109:12]
```

```
[@237,14:14='0',<const_interger>,109:14]
[@238,16:16=';',<';'>,109:16]
[@239,1:5='test2',<Identifier>,110:1]
[@240,6:7='->',<'->'>,110:6]
[@241,8:11='num1',<Identifier>,110:8]
[@242,13:13='=',<'='>,110:13]
[@243,15:15='0',<const_interger>,110:15]
[@244,17:17=';',<';'>,110:17]
[@245,1:1='i',<Identifier>,111:1]
[@246,2:3='++',<'++'>,111:2]
[@247,5:5=';',<';'>,111:5]
[@248,1:1='i',<Identifier>,112:1]
[@249,2:3='--',<'--'>,112:2]
[@250,5:5=';',<';'>,112:5]
[@251,1:1='i',<Identifier>,113:1]
[@252,3:3='=',<'='>,113:3]
[@253,5:5='1',<const_interger>,113:5]
[@254,7:7='+',<'+'>,113:7]
[@255,9:9='1',<const_interger>,113:9]
[@256,11:11=';',<';'>,113:11]
[@257,1:1='i',<Identifier>,114:1]
[@258,3:3='=',<'='>,114:3]
[@259,5:5='1',<const_interger>,114:5]
[@260,7:7='-',<'-'>,114:7]
[@261,9:9='1',<const_interger>,114:9]
[@262,11:11=';',<';'>,114:11]
[@263,1:1='i',<Identifier>,115:1]
[@264,3:3='=',<'='>,115:3]
[@265,5:5='1',<const_interger>,115:5]
[@266,7:7='*',<'*'>,115:7]
[@267,9:9='1',<const_interger>,115:9]
[@268,11:11=';',<';'>,115:11]
[@269,1:1='i',<Identifier>,116:1]
[@270,3:3='=',<'='>,116:3]
[@271,5:5='1',<const_interger>,116:5]
[@272,7:7='/',<'/'>,116:7]
[@273,9:9='1',<const_interger>,116:9]
[@274,11:11=';',<';'>,116:11]
[@275,1:1='i',<Identifier>,117:1]
[@276,3:3='=',<'='>,117:3]
[@277,5:5='1',<const_interger>,117:5]
[@278,7:7='%',<'%'>,117:7]
[@279,9:9='1',<const_interger>,117:9]
[@280,11:11=';',<';'>,117:11]
[@281,1:1='i',<Identifier>,118:1]
[@282,3:3='=',<'='>,118:3]
[@283,5:5='!',<'!'>,118:5]
[@284,6:6='i',<Identifier>,118:6]
[@285,8:8=';',<';'>,118:8]
[@286,1:1='i',<Identifier>,119:1]
[@287,3:3='=',<'='>,119:3]
[@288,5:5='i',<Identifier>,119:5]
```

```
[@289,7:7='&',<'&'>,119:7]
[@290,9:9='1',<const_interger>,119:9]
[@291,11:11=';',<';'>,119:11]
[@292,1:1='i',<Identifier>,120:1]
[@293,3:3='=',<'='>,120:3]
[@294,5:5='i',<Identifier>,120:5]
[@295,7:7='|',<'|'>,120:7]
[@296,9:9='1',<const_interger>,120:9]
[@297,11:11=';',<';'>,120:11]
[@298,1:1='i',<Identifier>,121:1]
[@299,3:3='=',<'='>,121:3]
[@300,5:5='i',<Identifier>,121:5]
[@301,7:8='&&',<'&&'>,121:7]
[@302,10:10='1',<const_interger>,121:10]
[@303,12:12=';',<';'>,121:12]
[@304,1:1='i',<Identifier>,122:1]
[@305,3:3='=',<'='>,122:3]
[@306,5:5='i',<Identifier>,122:5]
[@307,7:8='||',<'||'>,122:7]
[@308,10:10='1',<const_interger>,122:10]
[@309,12:12=';',<';'>,122:12]
[@310,1:1='i',<Identifier>,123:1]
[@311,3:3='=',<'='>,123:3]
[@312,5:5='i',<Identifier>,123:5]
[@313,7:7='^',<'^'>,123:7]
[@314,9:9='1',<const_interger>,123:9]
[@315,11:11=';',<';'>,123:11]
[@316,1:1='i',<Identifier>,124:1]
[@317,3:3='=',<'='>,124:3]
[@318,5:5='i',<Identifier>,124:5]
[@319,7:8='>>',<'>>'>,124:7]
[@320,10:10='1',<const_interger>,124:10]
[@321,12:12=';',<';'>,124:12]
[@322,1:1='i',<Identifier>,125:1]
[@323,3:3='=',<'='>,125:3]
[@324,5:5='i',<Identifier>,125:5]
[@325,7:8='<<',<'<<'>,125:7]
[@326,10:10='1',<const_interger>,125:10]
[@327,12:12=';',<';'>,125:12]
[@328,1:1='i',<Identifier>,126:1]
[@329,3:3='=',<'='>,126:3]
[@330,5:5='i',<Identifier>,126:5]
[@331,7:7='>',<'>'>,126:7]
[@332,9:9='1',<const_interger>,126:9]
[@333,11:11='?',<'?'>,126:11]
[@334,13:13='1',<const_interger>,126:13]
[@335,15:15=':',<':'>,126:15]
[@336,17:17='2',<const_interger>,126:17]
[@337,19:19=';',<';'>,126:19]
[@338,1:1='i',<Identifier>,127:1]
[@339,3:4='+=',<'+='>,127:3]
[@340,6:6='1',<const_interger>,127:6]
```

```
[@341,8:8=';',<';'>,127:8]
[@342,1:1='i',<Identifier>,128:1]
[@343,3:4='-=',<'-='>,128:3]
[@344,6:6='1',<const_interger>,128:6]
[@345,8:8=';',<';'>,128:8]
[@346,1:1='i',<Identifier>,129:1]
[@347,3:4='*=',<'*='>,129:3]
[@348,6:6='1',<const_interger>,129:6]
[@349,8:8=';',<';'>,129:8]
[@350,1:1='i',<Identifier>,130:1]
[@351,3:3='/',<'/'>,130:3]
[@352,4:4='=',<'='>,130:4]
[@353,6:6='1',<const_interger>,130:6]
[@354,8:8=';',<';'>,130:8]
[@355,1:1='i',<Identifier>,131:1]
[@356,3:4='%=',<'%='>,131:3]
[@357,6:6='1',<const_interger>,131:6]
[@358,8:8=';',<';'>,131:8]
[@359,1:1='i',<Identifier>,132:1]
[@360,3:4='&=',<'&='>,132:3]
[@361,6:6='1',<const_interger>,132:6]
[@362,8:8=';',<';'>,132:8]
[@363,1:1='i',<Identifier>,133:1]
[@364,3:4='|=',<'|='>,133:3]
[@365,6:6='1',<const_interger>,133:6]
[@366,8:8=';',<';'>,133:8]
[@367,1:1='i',<Identifier>,134:1]
[@368,3:4='^=',<'^='>,134:3]
[@369,6:6='1',<const_interger>,134:6]
[@370,8:8=';',<';'>,134:8]
[@371,1:1='i',<Identifier>,135:1]
[@372,3:5='>>=',<'>>='>,135:3]
[@373,7:7='1',<const_interger>,135:7]
[@374,9:9=';',<';'>,135:9]
[@375,1:1='i',<Identifier>,136:1]
[@376,3:5='<<=',<'<<='>,136:3]
[@377,7:7='1',<const_interger>,136:7]
[@378,9:9=';',<';'>,136:9]
[@379,1:2='if',<keyword>,137:1]
[@380,3:3='(',<'('>,137:3]
[@381,5:5='i',<Identifier>,137:5]
[@382,6:7='==',<'=='>,137:6]
[@383,8:8='1',<const_interger>,137:8]
[@384,10:10=')',<')'>,137:10]
[@385,11:11='{',<'{'>,137:11]
[@386,12:12='}',<'}'>,137:12]
[@387,1:2='if',<keyword>,138:1]
[@388,3:3='(',<'('>,138:3]
[@389,5:5='i',<Identifier>,138:5]
[@390,6:7='!=',<'!='>,138:6]
[@391,8:8='1',<const_interger>,138:8]
[@392,10:10=')',<')'>,138:10]
```

```
[@393,11:11='{',<'{'>,138:11]
[@394,12:12='}',<'}'>,138:12]
[@395,1:2='if',<keyword>,139:1]
[@396,3:3='(',<'('>,139:3]
[@397,5:5='i',<Identifier>,139:5]
[@398,6:6='>',<'>'>,139:6]
[@399,7:7='1',<const_interger>,139:7]
[@400,9:9=')',<')'>,139:9]
[@401,10:10='{',<'{'>,139:10]
[@402,11:11='}',<'}'>,139:11]
[@403,1:2='if',<keyword>,140:1]
[@404,3:3='(',<'('>,140:3]
[@405,5:5='i',<Identifier>,140:5]
[@406,6:6='<',<'<'>,140:6]
[@407,7:7='1',<const_interger>,140:7]
[@408,9:9=')',<')'>,140:9]
[@409,10:10='{',<'{'>,140:10]
[@410,11:11='}',<'}'>,140:11]
[@411,1:2='if',<keyword>,141:1]
[@412,3:3='(',<'('>,141:3]
[@413,5:5='i',<Identifier>,141:5]
[@414,6:7='>=',<'>='>,141:6]
[@415,8:8='1',<const_interger>,141:8]
[@416,10:10=')',<')'>,141:10]
[@417,11:11='{',<'{'>,141:11]
[@418,12:12='}',<'}'>,141:12]
[@419,1:2='if',<keyword>,142:1]
[@420,3:3='(',<'('>,142:3]
[@421,5:5='i',<Identifier>,142:5]
[@422,6:7='<=',<'<='>,142:6]
[@423,8:8='1',<const_interger>,142:8]
[@424,10:10=')',<')'>,142:10]
[@425,11:11='{',<'{'>,142:11]
[@426,12:12='}',<'}'>,142:12]
[@427,0:7='gotoFlag',<Identifier>,143:0]
[@428,8:8=':',<':'>,143:8]
[@429,1:6='return',<keyword>,144:1]
[@430,8:8='0',<const_interger>,144:8]
[@431,10:10=';',<';'>,144:10]
[@432,0:0='}',<'}'>,145:0]
[@433,0:3='void',<keyword>,146:0]
[@434,5:12='function',<Identifier>,146:5]
[@435,13:13='(',<'('>,146:13]
[@436,15:17='int',<keyword>,146:15]
[@437,19:22='arg1',<Identifier>,146:19]
[@438,24:24=',',<','>,146:24]
[@439,26:28='int',<keyword>,146:26]
[@440,30:33='arg2',<Identifier>,146:30]
[@441,35:35=')',<')'>,146:35]
[@442,0:0='{',<'{'>,147:0]
[@443,0:0='}',<'}'>,148:0]
```

上图显示，识别出了444个单词，识别基本正确。

6. 实验感想

在这次的实验过程中，我直接利用正则表达式进行书写Flex的.l文件生成，一方面我是对正则表达式用了更深的理解和运用，另一方面，我对C语言的规范也有了进一步的认识，虽然某些规则看起来很简单，但是在书写的时候要考虑到各种细节。不足的是，虽然利用Flex可以很好的匹配相应类别的单词，但缺少了错误处理的过程，在词法分析阶段，有些错误可以处理的可能要推迟到语法分析阶段，这样以来，编译的性能可能会有所减弱。

除此之外，我也理解了计卫星老师所给的BIT-minCC的框架，学会了如何在里面嵌入自己的代码和可执行程序，对整个编译过程有了更为熟悉的认识，要较好的实现编译器的前端，并不是一件容易的事。

对词法分析的设计与实现有了深刻的体会，尤其DFA的设计层面，如何完成单词的识别，如何对整个DFA进行汇总，刚开始我设计的DFA有很多缺陷不足，需要不断地修改，尽管现在也不是很完美，但是基本的识别功能已经完成。