# Documentation for Lexical Analyzer

Compiler Design (COMP 442/6421) Winter 2023

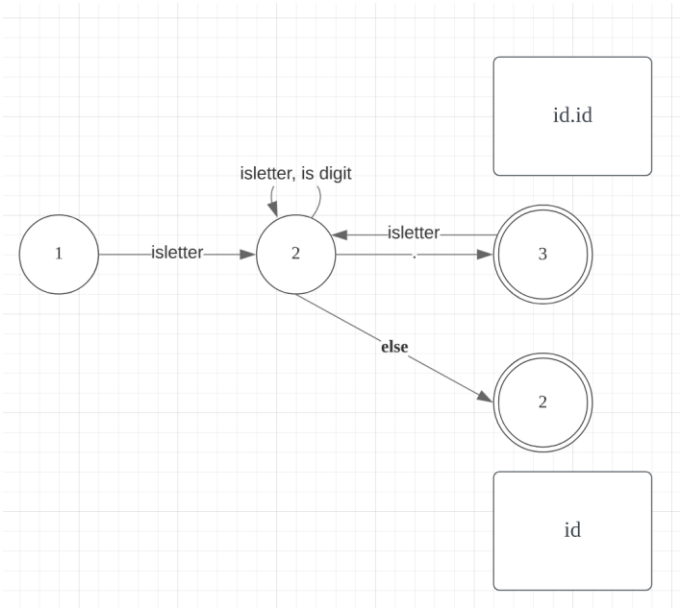
-Qiantongzhou(40081938)

**Document**

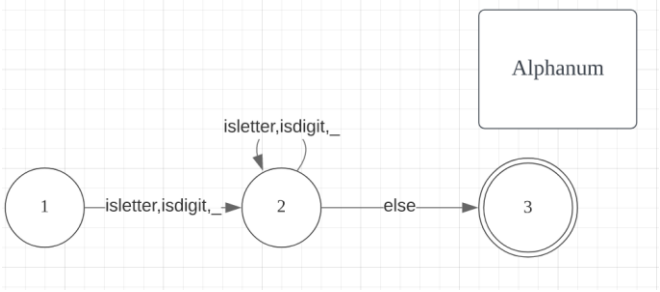
# Section 1. Lexical specifications

Regular expressions:

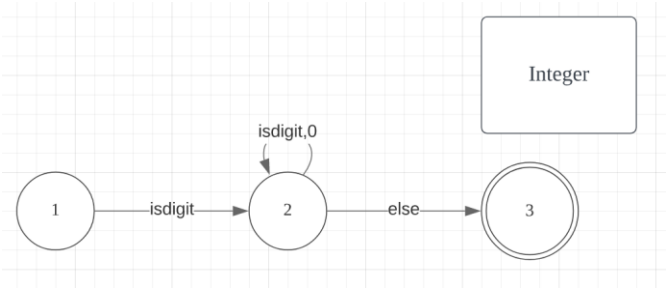
Id ::=  letter(letter|digit|_)*



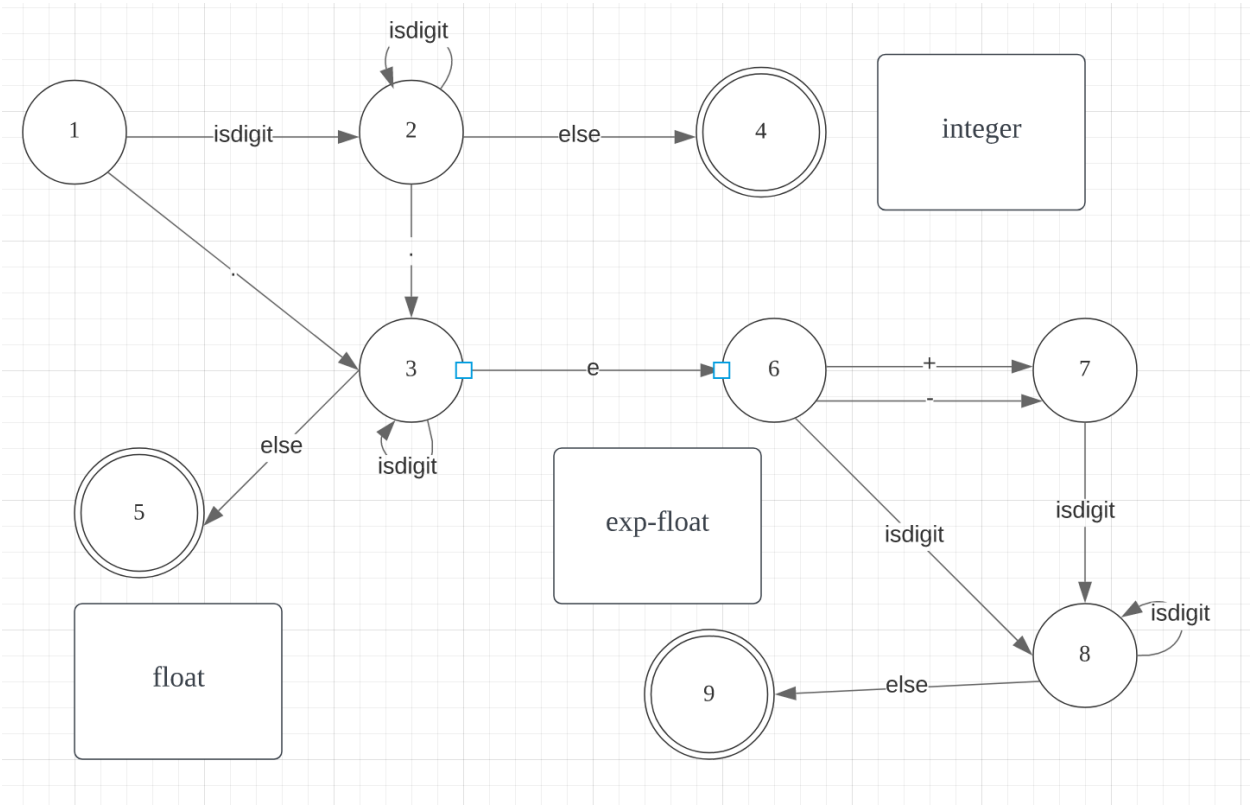

Alphanum ::= (letter|digit|_)(letter|digit|_)*



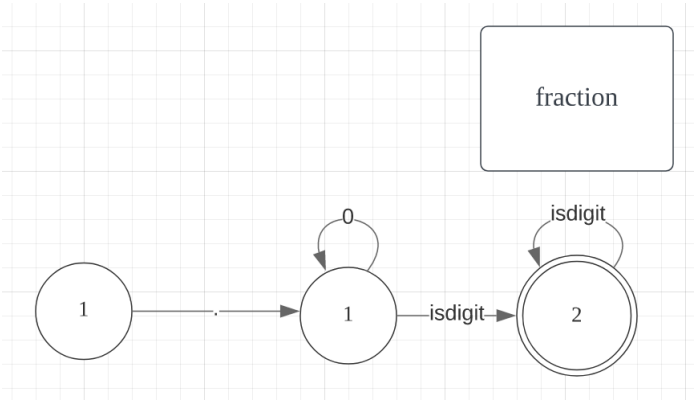
Integer ::= nonzero (digit* | 0)



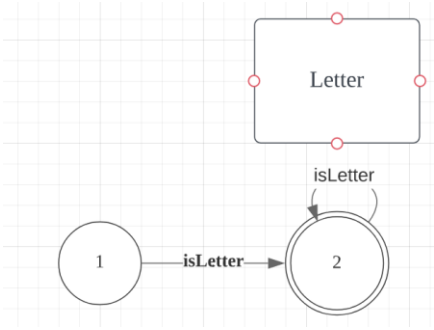

Float ::= (integer)(fraction)(e(+|-) | (integer)*)



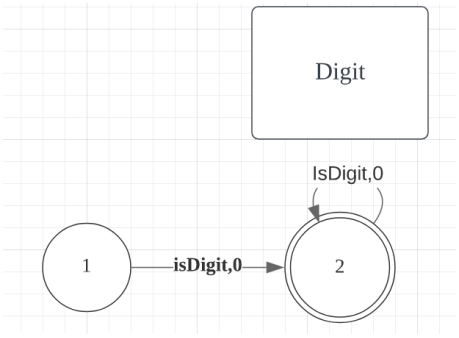

Fraction ::= .digit* nonzero |.0



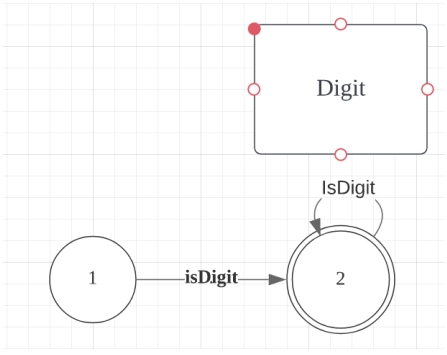
Letter ::= [a…z]|[A..Z]




Digit ::= [0..9]

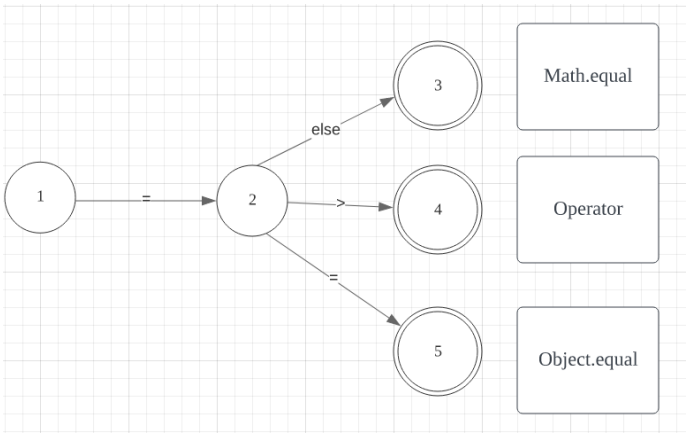



Nonzero ::= [1..9]

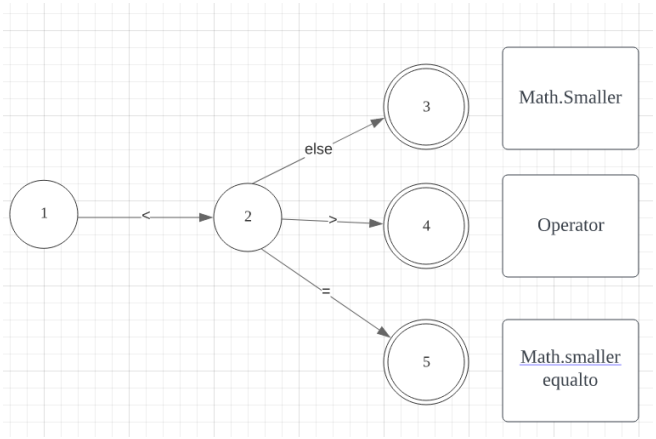



Regular expressions for signs

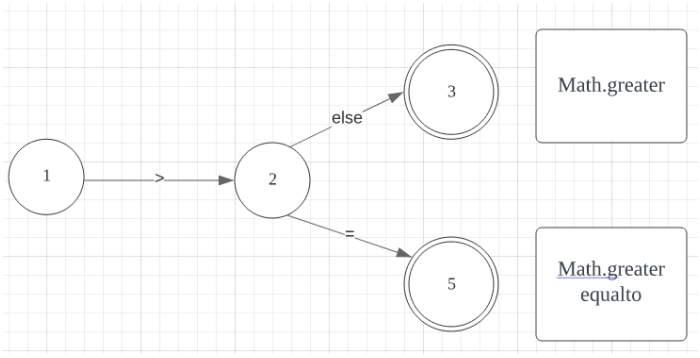
=(=|>)


1
=
2
else
>
=
3
4
5
Math.equal
Operator
Object.equal

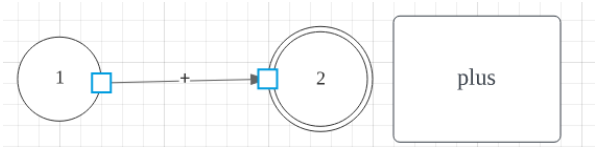
<(>|=)


1
<
2
else
>
=
3
4
5
Math.Smaller
Operator
Math.smaller
equalto

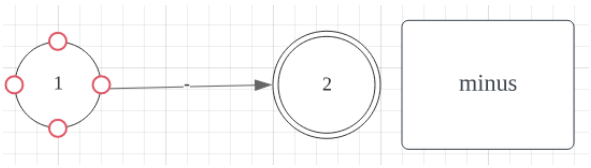
>(=|  )


1
>
2
else
=
3
5
Math.greater
Math.greater
equalto

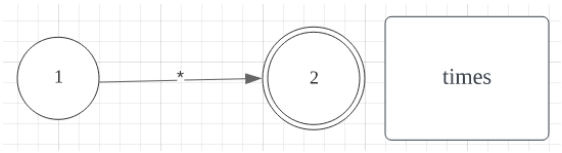
+


1
+
2
plus

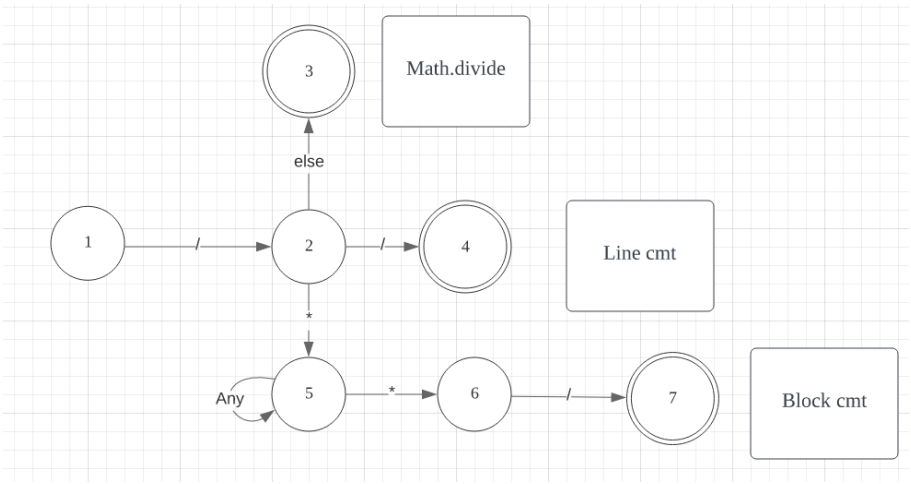
-




*




/(/|**/)




( any )

{ any }

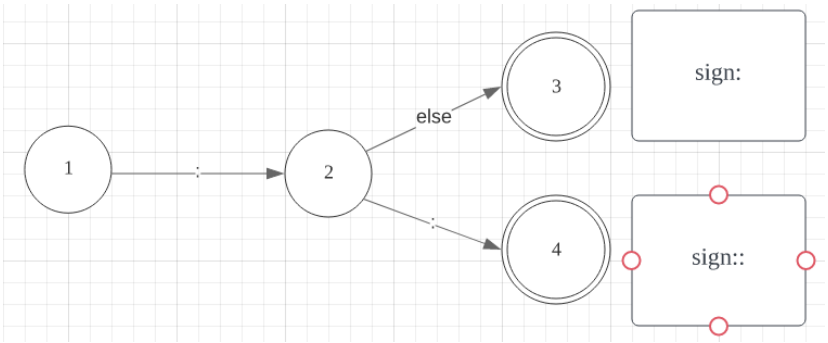
[ any ]

;

,

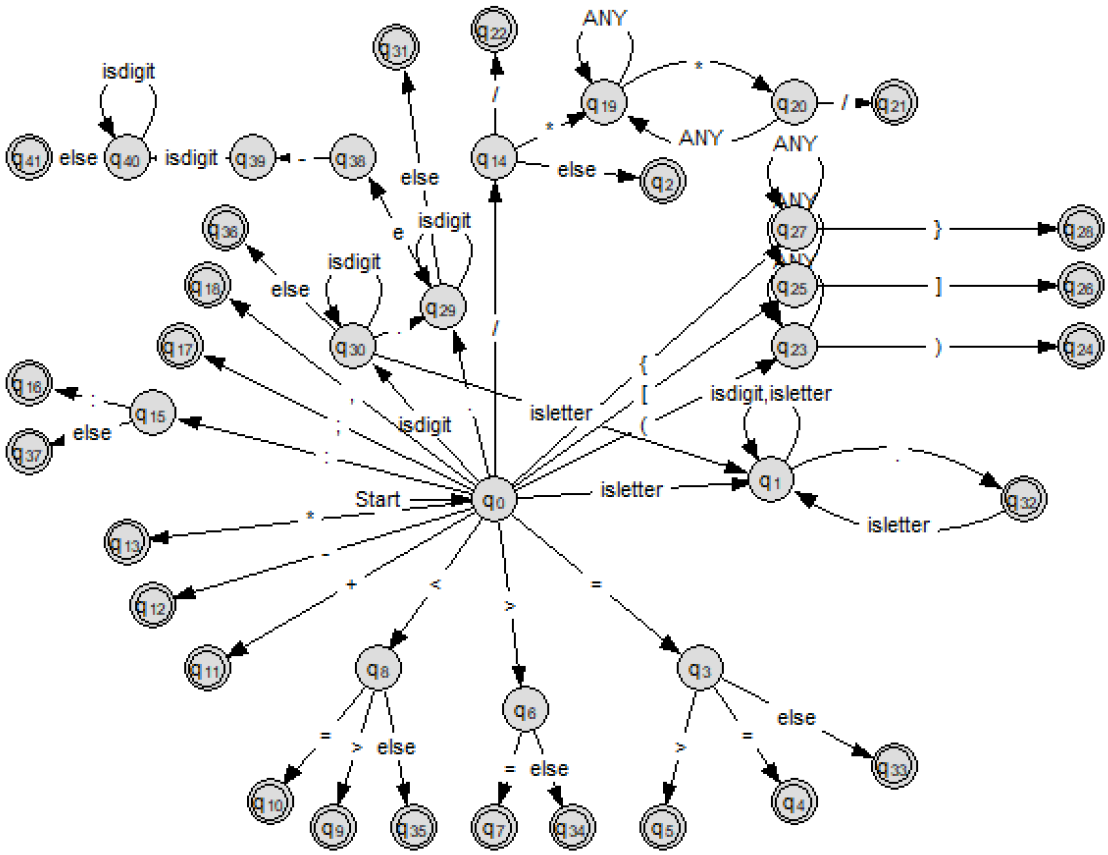
.

:(:|none)




Section 2:

## DFA for above:



# Transition Table

```
//q0            0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15    16       17   18 19
//                (  )  *  +  ,  -  .  /  :  ;  <  =  >  [  ] ANY isdigit isletter {  }
new List<int>(){ 0, 0,13,11,18,12,29,14,15,17, 8, 3, 6, 0, 0, 0,   30,       1,   0, 0 },
//q1
new List<int>(){ 0, 0, 0, 0, 0, 0,32, 0, 0, 0, 0 ,0 ,0, 0, 0, 0,    1,       1,   0, 0 },
//q2
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q3
new List<int>(){33,33,33,33,33,33,33,33,33,33,33, 4, 5,33,33,33,   33,      33,  33,33 },
//q4,f,==
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q5,f,=>
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q6
new List<int>(){34,34,34,34,34,34,34,34,34,34,34, 7,34,34,34,34,   34,      34,  34,34 },
//q7,f,>=
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q8
new List<int>(){35,35,35,35,35,35,35,35,35,35,35,10, 9,35,35,35,   35,      35,  35,35 },
//q9,f,<>
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q10,f,<=
//                (  )  *  +  ,  -  .  /  :  ;  <  =  >  [  ] ANY isdigit isletter {  }
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q11,f,+
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q12,f,-
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q13,f,*
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q14
new List<int>(){ 2, 2,19, 2, 2, 2, 2,22, 2, 2, 2, 2, 2, 2, 2, 2,    2,       0,   0, 0 },
//q15
new List<int>(){37,37,37,37,37,37,37,37,16,37,37,37,37,37,37,37,   37,      37,  37,37 },
//q16,f,::
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q17,f,;
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q18,f,,
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q19
new List<int>(){ 0, 0,20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,   19,      19,   0, 0 },
//q20
new List<int>(){ 0, 0, 0, 0, 0, 0, 0,21, 0, 0, 0, 0, 0, 0, 0, 0,   19,      19,   0, 0 },
//q21,f,/**/
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q22,f,//
//                (  )  *  +  ,  -  .  /  :  ;  <  =  >  [  ] ANY isdigit isletter {  }
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q23
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q24
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q25
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q26
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q27
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q28
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q29
new List<int>(){31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,   29,      38,  31,31 },
//q30,f,digit
new List<int>(){36,36,36,36,36,36,29,36,36,36,36,36,36,36,36,36,   30,       1,  36,36 },
//q31,f,float
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q32
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       1,   0, 0 },
//q33
```

```
 //             (  )  *  +  ,  -  .  /  :  ;  <  =  >  [  ] ANY isdigit isletter {  }
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q34
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q35
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q36
new List<int>(){36,36,36,36,36,36,36,36,36,36,36,36,36,36,36,36,   36,      36,  36,36 },
//q37
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
//q38
new List<int>(){ 0, 0, 0,39, 0,39, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,   39,       0,   0, 0 },
//q39
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,   40,       0,   0, 0 },
//q40
new List<int>(){41, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,   40,       0,   0, 0 },
//q41
new List<int>(){ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0,       0,   0, 0 },
```

# Reserved words:

"integer","float","void","class","self","isa","while","if","then","else",
"read","write","return","localvar","constructor","attribute","function","public",
"private"

Section 3:

**Check words**

```
private bool isReservedword(string word) {
    //no reserved word more than 16
    if((word).Length > 16 || (word).Length == 0)
        return false;
    //List check which one
    var Reservedwords = new List<string>(){
        "integer","float","void","class","self","isa","while","if","then","else",
    "read","write","return","localvar","constructor","attribute","function","public","private"};
    // Lambda
    return Reservedwords.Exists(element => (word.ToLower()) == element);
}
```

Every time a Char is added to token, dfa check if this is a reserved word.

**Get next state form transition table**

```
//Get next state from transition table
CurrentState = getNextState(CurrentState,cChar);
```

Check if the sign leads to which state

If not in the transition table, return to 0 state and report error

**Error for cmt, do not process cmt**

```
if(cChar == '/' && CharTemp == '*') {
    if(code.Length - 1 == Index)
        return Output;
    CharTemp = code[++Index];
    //remove everything till see */
    do {
        cChar = CharTemp;
        if(code.Length - 1 == Index)
            return Output + cChar;
        CharTemp = code[++Index];
        //without */ error
        if (code.Length - 1 == Index)
        {
            Console.WriteLine("Error: at line:" +
            outlexerrors[Tokens.Punctuation].Add('
            return Output;
        }
    } while(cChar != '*' && CharTemp != '/');
    Output += '\r';
    if(code.Length - 1 != Index)
        CharTemp = code[++Index];
    continue;
}
```

If the system find /*, Goes to state forward character until see the sign */, else print cmt error

## Error Detection

```
if (char.IsDigit(CurrentToken[0]))
{
    Console.WriteLine("Error: at line:" +
    outlexerrors[Tokens.id].Add("Error: a
}else
if (CurrentToken.Substring(0, 1) == "_")
{
    Console.WriteLine("Error: at line:" +
    outlexerrors[Tokens.id].Add("Error: a
}else
if (!char.IsLetter(CurrentToken[0]))
{
    Console.WriteLine("Error: at line:" +
    outlexerrors[Tokens.id].Add("Error: a

}
```

Only three type of error, identification error, integer error and float error

Error is going to be detected before put into token list in case switch

# Use of Tool

## C# Section 4

Build in tool in C#

```
char.IsDigit(cChar)
```

```
(char.IsLetter(cChar)
```

This two function return if the character is digit or letter.

Help us get the correct transition state in DFA.

Problems:

Unable to detect 0, need a extra tool to spilt 0 from string

Unable to detect _, need the scan identitfication again to locate problems

```
outlexerrors = new Dictionary<Tokens, Dictionary<string, string>>();
```

Use of dictionary, label errors and type for later recovery

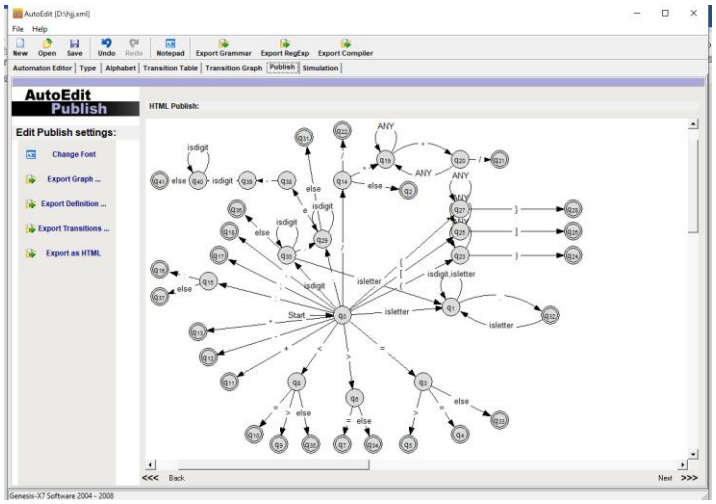
```
CurrentToken.Substring(0, 1) == "0"
```

Use of string operation to detect Errors.

```
using (StreamWriter writer1 = new StreamWriter(dir + "OutputErrors/"+"outlexerrors_"+file))
{
    writer1.WriteLine("This is all the lexerrors");

    foreach (var type in outlexerrors.Keys)
    {
        writer1.WriteLine("\n\n" + type.ToString() + ":\n");
        foreach (var token in outlexerrors[type])
        {

            //Console.WriteLine(token.Key);
            writer1.WriteLine(token.Key);
        }

    };
}
```

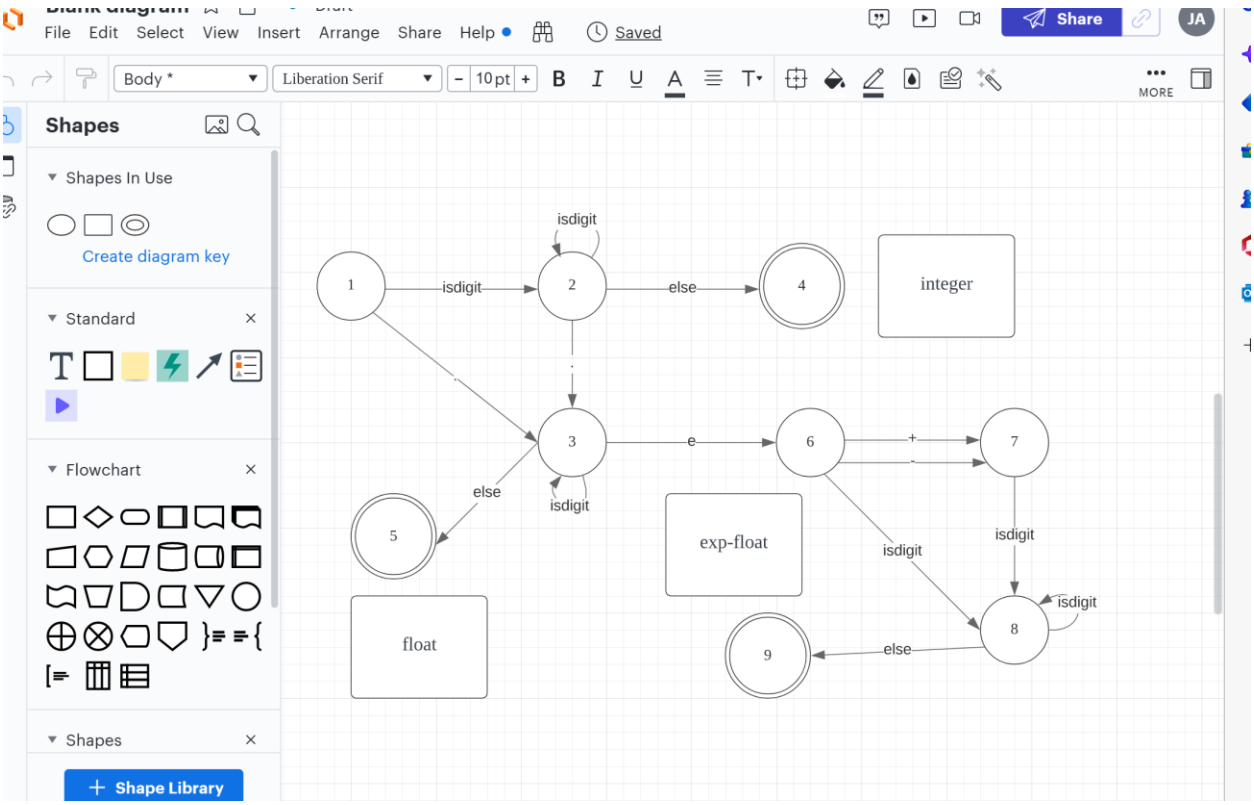
File handler to write files to folder.

# AutoEdit




Help to get the transition table for the whole DFA.

**Lucidchart**




**For prototyping a DFA for each cases**

**Later used for create AutoEdit chart**