**Lexical Analysis**

```
main()
{
        int i;
        i=105;
}
```

This code is a string which looks like :

main()\n{\n \t int i; \n \t int i=105; \n} EOF

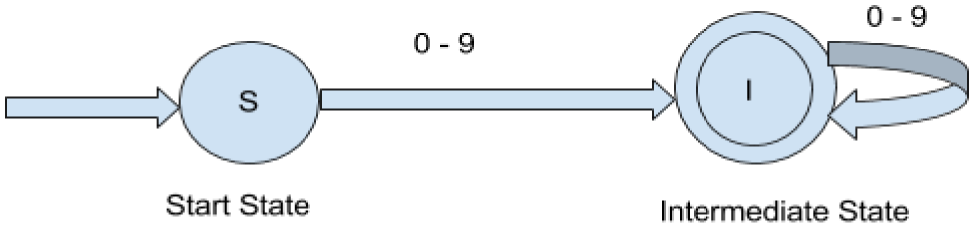
(1D not 2D)

A regex for an integer looks like

[0-9]* ≡ [0-9][0-9]*

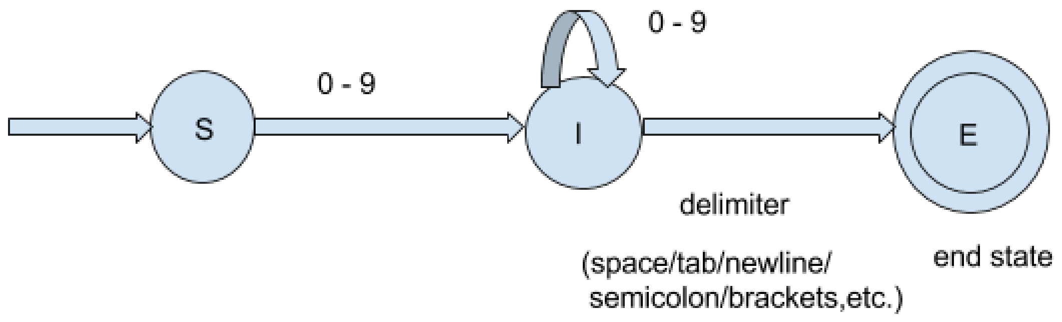
Regex → NFA → DFA → minimum state DFA

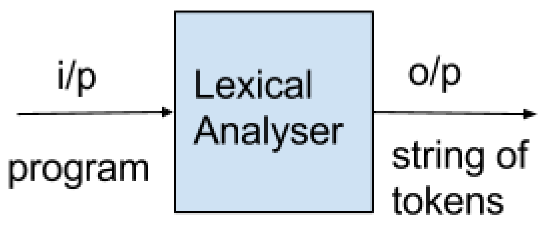
So the language of integers can be represented as



But we need to know when to stop

So recognition of integers requires a slight change to the language





<u>PROBLEMS:</u>

1. We have lost the delimiter. We need to store it.
   eg. ’:’ semicolon should not be lost. Therefore replace the delimiter back.
2. We need to remember the value of the integer. Before going to state I .

<u>State S :</u>

```
        c=getchar(ifp);                         { ifp : input file pointer }
        if(c >='0' && c<='9')
        {
                var = c - '0';
                goto state I;
        }
```

State I :

```
c = getchar(ifp);
if( c<=’9’ && c>=’0’)
{
        val=val x 10 + c - ‘0’ ;
        goto state I;
}
else if( delimiter(c) == 0)
{
        unget(c,ifp);
        goto state E;
}
else
{
        printf(“Error : Invalid int”);
        unget(c,ifp);
        goto State S;
}
```

State E :

```
s.value = val;                    {s is a token}
s.type = INTEGER;
fprintf( ofp,s );   {ofp is the output file pointer}
goto state S;
```

When we reach end state E, we know that we have recognised an integer.

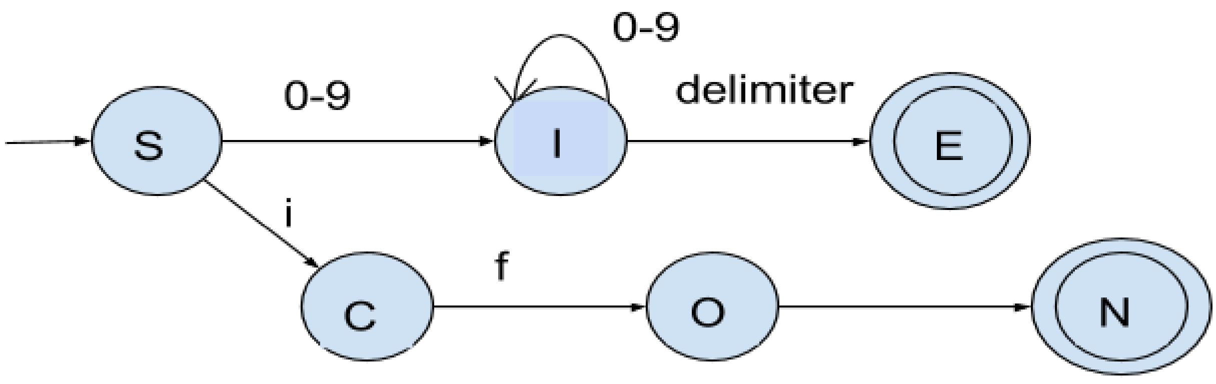
Each file is processed into series of ‘lexens’ (tokens).
Each statement becomes a label and transitions among states depends on the input.

eg. To recognise ‘if’ condition :-

```
State S :-    c = getchar( ifp);
              if(c >='0' && c<='9')
              {
                      var = c - '0';
                      goto state I;
              }
              else if( c == 'i')
              {
               goto state C;
              }

State C :-    c = getchar( ifp);
              if( c == 'f')
              {
               goto state N;
              }

State N :-    c = getchar( ifp);
              if( delimiter(c) == 0 )
              {
                      fprintf( ofp, IF COND);
                      ungetc(ifp);
              }
```



if keyword

Overloaded Symbols :

eg. = or ==
    > or >=
    <  or <=

When we see ‘>’ we don’t know if it is a ‘>’ or ‘>=’. We need to do a lookahead to find out the exact symbol/token.

Rule in Lexical Analysis : The longest string that matches the symbol is a valid token.