ASSIGNMENT I

1. Write a program to find out whether given input is a letter or digit.

Solution: lex1.l

```
%{


%}
letter [a-zA-Z]
digit [0-9]
id2 {letter}({letter}|{digit})*
num {digit}("."({digit})+)?

%%
"if"|"else"|"while"|"for"  {printf("keyword");}
{num} {printf("num");}
{id2}    { printf("id2 "); }
%%

int main()
{
yylex();
return 0;
}
```

Execution:

```
1. flex lex1.l
2. cc lex.yy.c -lfl
3. ./a.out
```

2. Write a program to find out whether given input is a noun, pronoun,verb, adverb, adjective or preposition

Solution: lex2.l

```
%{

/*This sample demonstrates a word as a verb/ not a verb */

%}


%%

[\t]+              /*Ignore whitespaces*/;

is|

am |

are |

is|

were |

was |

be|

being |

been |

do|

does |

did|

will|

would|

should|
can|

could|

has|

have|

had|

go   {printf("%s: is a verb\n",yytext); }

[a-zA-Z]+  {printf("%s: is not a  verb\n",yytext); }

.|\n       { ECHO:}

%%

main()

{

yylex();

}
```

Execution:

```
1. flex lex.l
2. cc lex.yy.c -lfl
3. ./a.out
```

Note : Extend this program to include noun, pronoun, adverb,adjective or preposition.

ASSIGNMENT II

Problem Statement

Implement Lexical analyser for sample language using LEX with error handling. (Subset of C).

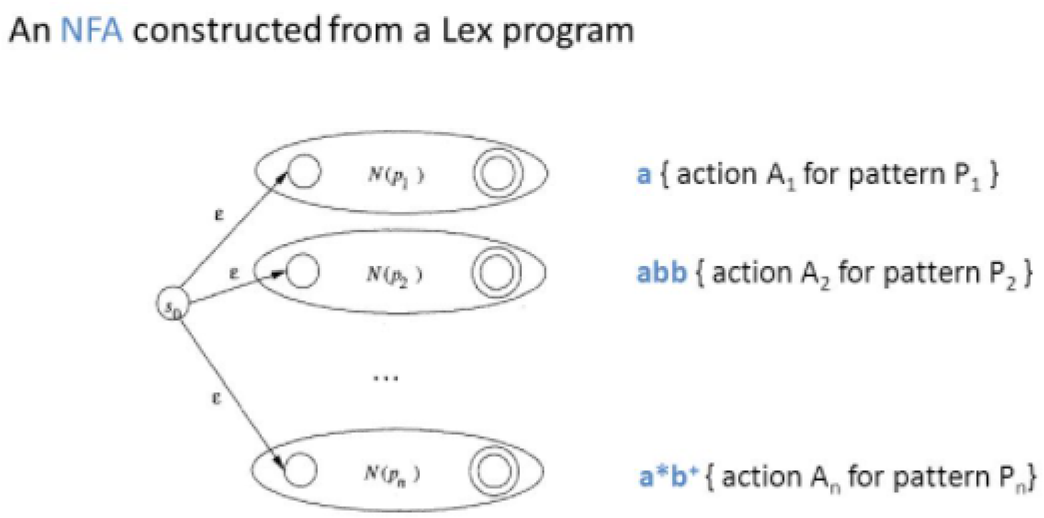
Objective

To understand how to build a Lexical Analyser.

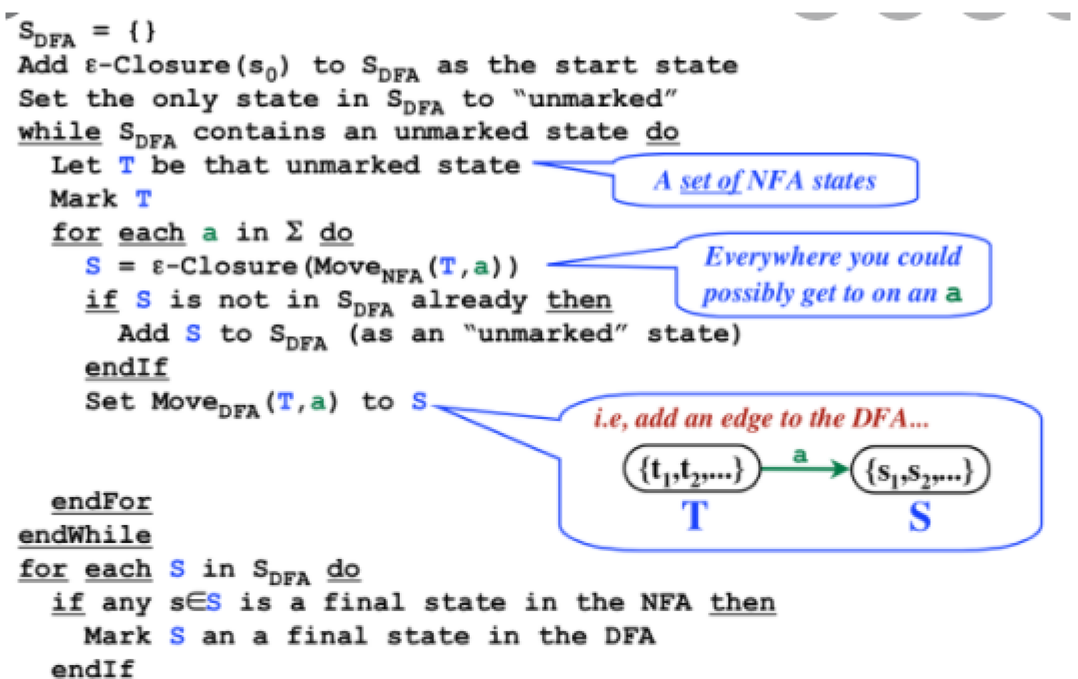
Theory

Step 1: Construct ε-NFA from the Regular Expressions

Step 2:


An NFA constructed from a Lex program
s_0
ε
ε
ε
$N(p_1)$
$N(p_2)$
...
$N(p_n)$
a { action A_1 for pattern P_1 }
abb { action A_2 for pattern P_2 }
a*b+ { action A_n for pattern P_n }

Convert ε-NFA to DFA using Subset Construction.


```
S_DFA = {}
Add ε-Closure(s_0) to S_DFA as the start state
Set the only state in S_DFA to "unmarked"
while S_DFA contains an unmarked state do
  Let T be that unmarked state
  Mark T
  for each a in Σ do
    S = ε-Closure(Move_NFA(T,a))
    if S is not in S_DFA already then
      Add S to S_DFA (as an "unmarked" state)
    endIf
    Set Move_DFA(T,a) to S

  endFor
endWhile
for each S in S_DFA do
  if any s∈S is a final state in the NFA then
    Mark S an a final state in the DFA
  endIf
```
A set of NFA states
Everywhere you could possibly get to on an a
i.e, add an edge to the DFA...
{t1,t2,...}
a
{s1,s2,...}
T
S

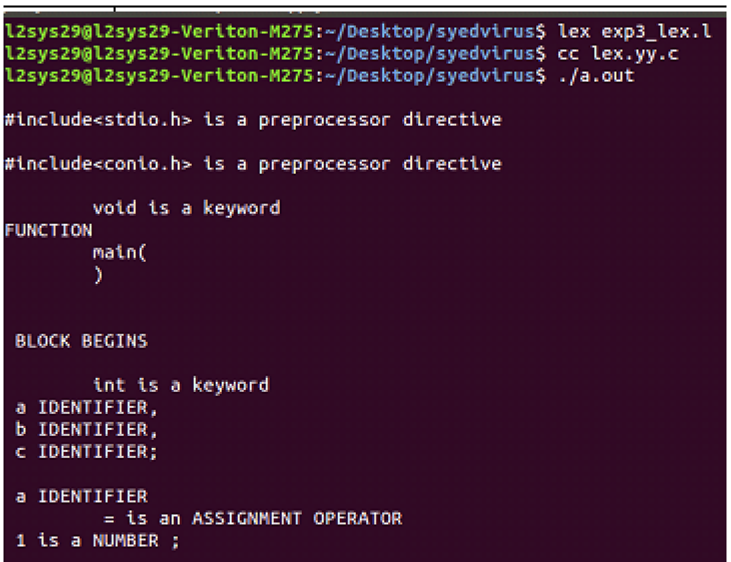
Solution:

```
//Implementation of Lexical Analyzer using Lex tool
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}

"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}

{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}

\{{if(!COMMENT)printf("\n BLOCK BEGINS");}

\}{if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc, char **argv)
{
FILE *file;
file=fopen("var.c","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

INPUT:

```
//var.c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

OUTPUT:



```
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ lex exp3_lex.l
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ cc lex.yy.c
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out

#include<stdio.h> is a preprocessor directive

#include<conio.h> is a preprocessor directive

        void is a keyword
FUNCTION
        main(
        )


 BLOCK BEGINS

        int is a keyword
 a IDENTIFIER,
 b IDENTIFIER,
 c IDENTIFIER;

 a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 1 is a NUMBER ;
```