Week-4

4. Implementation of lexical analyzer using LEX

```
/* program name is lexp.l */
%{

int COMMENT=0;
int cnt=0;
%}

identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto {printf("\n\t%s is a KEYWORD",yytext);}
"/*" {COMMENT = 1;}
"*/" {COMMENT = 0; cnt++;}
{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n 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\t%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) {
        if (argc > 1) {
                FILE *file;
                file = fopen(argv[1],"r");
        if(!file) {
                printf("could not open %s \n",argv[1]);
                exit(0);
        }
        yyin = file;
        }
        yylex();
        printf("\n\n Total No.Of comments are %d",cnt);
        return 0;
}

int yywrap() {
        return 1;
}
```

Output:

```
C:\Users\Yeswanth\Downloads\Sem5\Compilers\CD-Lex-Programs>flex lextool.l

C:\Users\Yeswanth\Downloads\Sem5\Compilers\CD-Lex-Programs>gcc lex.yy.c

C:\Users\Yeswanth\Downloads\Sem5\Compilers\CD-Lex-Programs>a.exe
  #include<stdio.h>

#include<stdio.h> is a Preprocess directive
main()


Function
        main(
        )

{

 Block begins
int a,b;

        int is a Keyword
 a Identifier,
 b Identifier;
}

 Block ends
```