5. Create lexical analyzer in lex

Program :

```
%{
int COMMENT=0;
char code[100];
%}

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%s is a keyword",yytext);}

"/*" {COMMENT=1;}{printf("\n%s : comment",yytext);}
{identifier}\( {printf("\nFUNCTION \n%s",yytext);}
\{  {printf("\n BLOCK BEGINS");}
\}  {printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {printf("\n%s : identifier",yytext);}
\".*\" {printf("\n%s : string",yytext);}
[0-9]+ { printf("\n%s : number ",yytext);}
\)(\:)? {printf("\n");ECHO;printf("\n");}
\( ECHO;
= {printf("\n%s : assingment operator",yytext);}
\<= |
\>= |
\< |
== |
\> {printf("\n%s : relational operator",yytext);}
```

```
%%
int main(int argc, char **argv)
{
 printf("Enter Program: ");
 scanf("%s",code);
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

Output :

```
C:\GnuWin32\bin>a.exe
Enter Program:    n = 123

= : assingment operator
123 : number
int i = 0

int is a keyword
i : identifier
= : assingment operator
0 : number
while i < n

while is a keyword
i : identifier
< : relational operator
n : identifier
}
```