Programming Assignment-2 - Implementation of Lexical Analyzer using LEX tool

HARSHINI S
185001058

Program Code:

```
%{
 #include
 %}
anyChar .|\n
multiline \/\*{anyChar}*\*\/
single \/\/.*
dir #.*
str \".*\"
c \'.\'
digit [0-9]
dec digit+\.digit+
hex [0-9A-Fa-f]+[hH]
arith [+\-*/%]
logical (&&)|(\|\|)|(!)
relational \<|\<=|\>|\>=|==|!=
bitwise (\^)|&|(\|)|(\<\<)|(\>\>)
unary \+\+|\-\-
special [;|,|\.|\[|\]|\{|\}|\(|\)]
%%
{multiline} {printf("%-30s \t\t- Multiline comment\n",yytext);}
{single} { printf("%-30s - Single Line comment\n", yytext); }
{dir} { printf("%-30s - preprocessor directive\n",yytext);}

auto|break|case|char|const|continue|default|do|double|else|enum|extern |float |
for|goto|if|int|long|register|return|short|signed|sizeof |
static|struct|switch|typedef|union|unsigned|void|volatile |

while {printf("%-30s - keyword\n",yytext);}
{str} { printf("%-30s - String const\n", yytext); }
{c} { printf("%-30s - Character const\n", yytext); }
{dec} { printf("%-30s - Double\n", yytext); }
{hex} { printf("%-30s - Hexadecimal Integer\n", yytext); }
{digit}+ { printf("%-30s - Decimal Integer\n", yytext); }
{arith}= { printf("%-30s - Arithmetic assignment operators\n", yytext); }
{arith} { printf("%-30s - Arithmetic operators\n", yytext); }
{logical} { printf("%-30s - Logical operator\n", yytext); }
{relational} { printf("%-30s - Relational operator\n", yytext); }
{bitwise} { printf("%-30s - Bitwise operator\n", yytext); }
```

```
{unary} { printf("%-30s - Unary operator\n", yytext); }
= { printf("%-30s - Assignment operator\n", yytext); }
{special} { printf("%-30s - Special Character\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*\(.*\) { printf("%-30s - Function call\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { printf("%-30s - Identifier\n", yytext); }
.|\n { }
%%
int main()
{
FILE *file;
file=fopen("sample.c","r");
if(!file)
{ printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
```