

# Compiler lab

Team members:106119097,106119107

Code:

```
Open  [?] lexicalAnalyz.l ~/Compiler/ Save
1
2 %{
3 #include<stdio.h>
4 #include<string.h>
5 int COMMENT=0;
6 }
7 digit [0-9]*
8 letter [a-zA-Z]*
9 identifier [a-zA-Z][a-zA-Z0-9]*
10 Integer (digit)+("E"|"+"|"-"?)?(digit)+?
11 Float (digit)+"."(digit)+("E"|"+"|"-"?)?(digit)+?
12 op [-+*/%]/[!|=]
13 wrongId [0-9][a-zA-Z0-9]*
14
15 %%
16 #.* {printf("\n%s is a preprocessor directive\n",yytext);}
17
18 ent |
19 flot |
20 chaar |
21 double |
22 wtle |
23 foor |
24 strct |
25 doo |
26 ef |
27 brk |
28 cnt |
29 void |
30 switch |
31 rtrn |
32 else |
33 printf {printf("\n%s is a keyword\n",yytext);}
34
35 [0-9]+ {printf("\n%s is a Integer Literal\n",yytext);}
36 [0-9]+\.[0-9]+ {printf("\n%s is a Float Literal\n",yytext);}
37 "[a-zA-Z0-9_]" {printf("\n%s is a word or letter\n",yytext);}
38 [letter] {printf("\n%s is a word or letter\n",yytext);}
39
40 "&&" {printf("\n%s is a logical AND operator\n",yytext);}
41 "<" {printf("\n%s is a less than operator\n",yytext);}
42 ">" {printf("\n%s is a greater than operator\n",yytext);}
43 "<=" {printf("\n%s is a less than or equal to operator\n",yytext);}
44 ">=" {printf("\n%s is a greater than or equal to operator\n",yytext);}
45 "==" {printf("\n%s is a equal to operator\n",yytext);}
46 "=" {printf("\n%s is a assignment operator\n",yytext);}
47 "+" {printf("\n%s is a addition operator\n",yytext);}
48 "-" {printf("\n%s is a subtraction operator\n",yytext);}
49 "*" {printf("\n%s is a multiplication operator\n",yytext);}
50 "/" {printf("\n%s is a divide operator\n",yytext);}
51 "%" {printf("\n%s is a modulus operator\n",yytext);}
52 "&" {printf("\n%s is a bitwise AND operator\n",yytext);}
53 "^" {printf("\n%s is a bitwise XOR operator\n",yytext);}
54 "|" {printf("\n%s is a bitwise OR operator\n",yytext);}
55
56 "(" {printf("\n%s is a open curly bracket\n",yytext);}
57 ")" {printf("\n%s is a close curly bracket\n",yytext);}
58 "[" {printf("\n%s is a open square bracket\n",yytext);}
59 "]" {printf("\n%s is a close square bracket\n",yytext);}
60 "{" {printf("\n%s is a open curly bracket\n",yytext);}
61 "}" {printf("\n%s is a close curly bracket\n",yytext);}
62 "\"" {printf("\n%s is a double quote\n",yytext);}
63 "\"" {printf("\n%s is a double quote\n",yytext);}
64 "\"\" {printf("\n%s is a back slash\n",yytext);}
65 ";" {printf("\n%s is a semi colon\n",yytext);}
66 "," {printf("\n%s is a comma\n",yytext);}
67
68
69
70 {wrongId} {printf("\n%s identifier cant start with number\n",yytext);}
71 {identifier} {printf("\n%s is a identifier\n",yytext);}
72 "%d"|"s"|"c"|"f"|"e" {printf("\n%s is a format specifier\n",yytext);}
73 %%
74
75 int yywrap(void){}
76
77 int main()
78 {
79     yylex();
80
81     return 0;
82 }
```

## Output:

```
methul@methul: ~/compiler
methul@methul:~/compiler$ flex lexicalAnalyze.l
methul@methul:~/compiler$ gcc lex.yy.c -ll
methul@methul:~/compiler$ ./a.out
ent k=50;flot x=58.69;

    ent is a keyword
=    k is a identifiers
    Assignment operator
;    50 is a Integer Literal
    SEMI COLON
    flot is a keyword
=    x is a identifiers
    Assignment operator
;    58.69 is a Float Literal
    SEMI COLON

if(k<sum){printf("hello");}else{printf("hi")}

(    if is a identifiers
    OPEN BRACKET
<    k is a identifiers
    GREATER THAN operator
)    sum is a identifiers
    CLOSE BRACKET
{    OPEN_CURLY BRACKET

    printf is a keyword
(    OPEN BRACKET
"    DOUBLE QUOTE
    hello is a identifiers
    DOUBLE QUOTE
)    CLOSE BRACKET
}    CLOSE_CURLY BRACKET

{    else is a keyword
    OPEN_CURLY BRACKET

    printf is a keyword
(    OPEN BRACKET
"    DOUBLE QUOTE
    hi is a identifiers
    DOUBLE QUOTE
)    CLOSE BRACKET
}    CLOSE_CURLY BRACKET

int 0sum=50;

    int is a identifiers

    0sum identifier cant start with number
=    Assignment operator
;    50 is a Integer Literal
    SEMI COLON
```