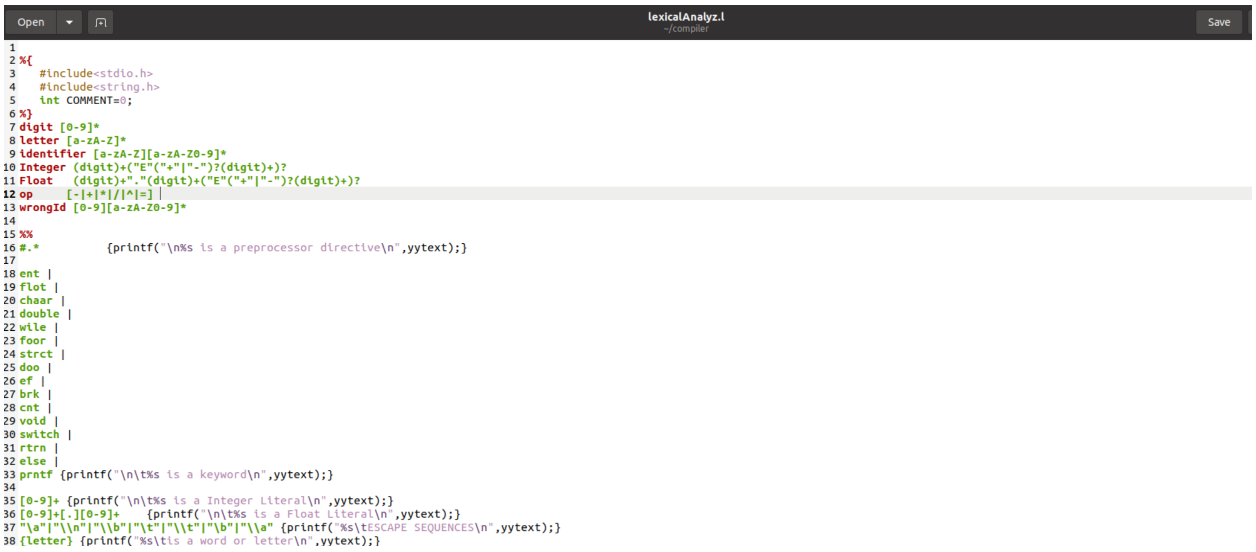
Compiler lab

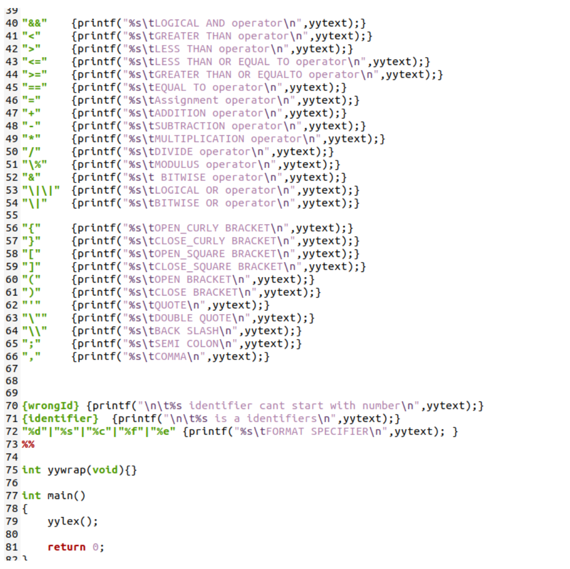
Team members:106119097,106119107

Code:



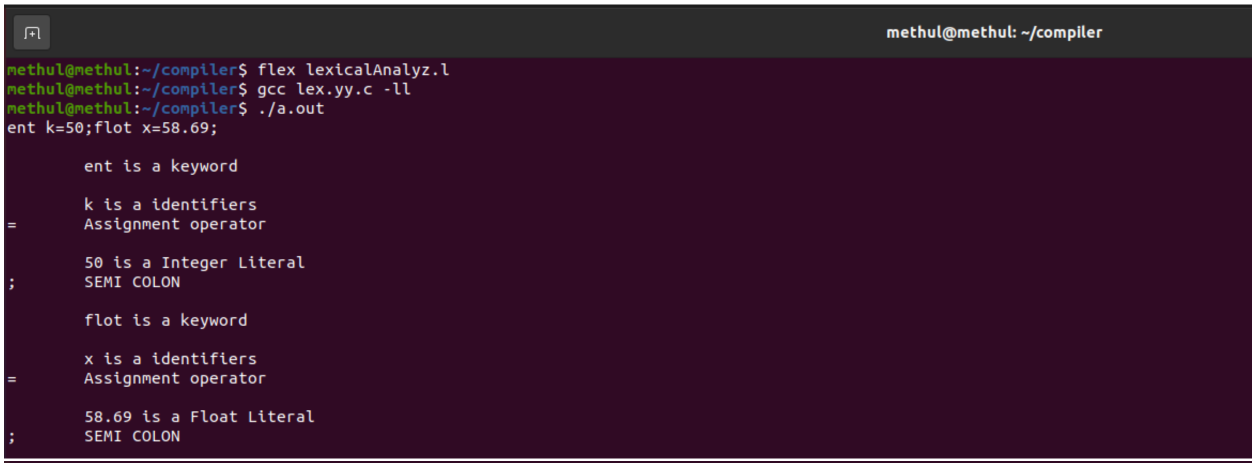
lexicalAnalyz.l
~/compiler

```

%{
   #include<stdio.h>
   #include<string.h>
   int COMMENT=0;
%}
digit [0-9]*
letter [a-zA-Z]*
identifier [a-zA-Z][a-zA-Z0-9]*
Integer (digit)+("E"("+"|"-")?(digit)+)?
Float   (digit)+"."(digit)+("E"("+"|"-")?(digit)+)?
op      [-|+|*|/|^|=]
wrongId [0-9][a-zA-Z0-9]*

%%
#.*          {printf("\n%s is a preprocessor directive\n",yytext);}

ent |
flot |
chaar |
double |
wile |
foor |
strct |
doo |
ef |
brk |
cnt |
void |
switch |
rtrn |
else |
prntf {printf("\n\t%s is a keyword\n",yytext);}

[0-9]+ {printf("\n\t%s is a Integer Literal\n",yytext);}
[0-9]+[.][0-9]+    {printf("\n\t%s is a Float Literal\n",yytext);}
"\a"|"\\n"|"\\b"|"\t"|"\\t"|"\b"|"\\a" {printf("%s\tESCAPE SEQUENCES\n",yytext);}
{letter} {printf("%s\tis a word or letter\n",yytext);}
```



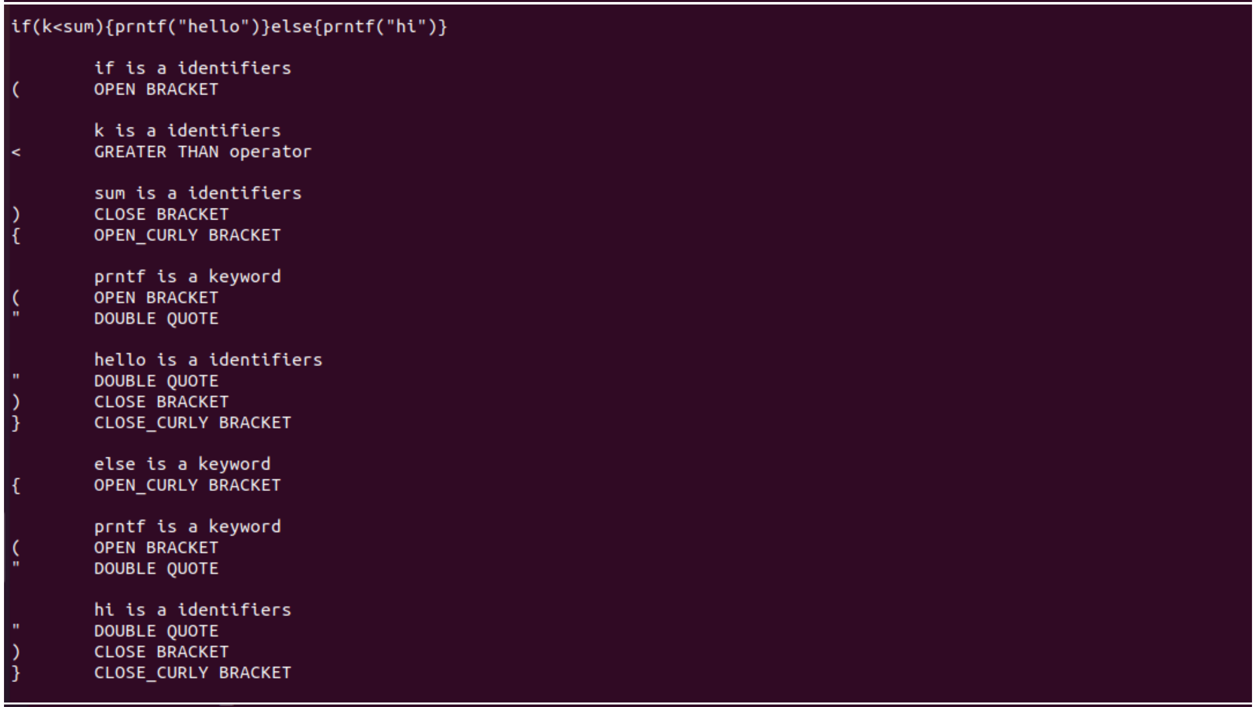
```

"&&"    {printf("%s\tLOGICAL AND operator\n",yytext);}
"<"     {printf("%s\tGREATER THAN operator\n",yytext);}
">"     {printf("%s\tLESS THAN operator\n",yytext);}
"<="    {printf("%s\tLESS THAN OR EQUAL TO operator\n",yytext);}
">="    {printf("%s\tGREATER THAN OR EQUALTO operator\n",yytext);}
"=="    {printf("%s\tEQUAL TO operator\n",yytext);}
"="     {printf("%s\tAssignment operator\n",yytext);}
"+"     {printf("%s\tADDITION operator\n",yytext);}
"-"     {printf("%s\tSUBTRACTION operator\n",yytext);}
"*"     {printf("%s\tMULTIPLICATION operator\n",yytext);}
"/"     {printf("%s\tDIVIDE operator\n",yytext);}
"\%"    {printf("%s\tMODULUS operator\n",yytext);}
"&"     {printf("%s\t BITWISE operator\n",yytext);}
"\|\|"  {printf("%s\tLOGICAL OR operator\n",yytext);}
"\|"    {printf("%s\tBITWISE OR operator\n",yytext);}

"{"     {printf("%s\tOPEN_CURLY BRACKET\n",yytext);}
"}"     {printf("%s\tCLOSE_CURLY BRACKET\n",yytext);}
"["     {printf("%s\tOPEN_SQUARE BRACKET\n",yytext);}
"]"     {printf("%s\tCLOSE_SQUARE BRACKET\n",yytext);}
"("     {printf("%s\tOPEN BRACKET\n",yytext);}
")"     {printf("%s\tCLOSE BRACKET\n",yytext);}
"'"     {printf("%s\tQUOTE\n",yytext);}
"\""    {printf("%s\tDOUBLE QUOTE\n",yytext);}
"\\"    {printf("%s\tBACK SLASH\n",yytext);}
";"     {printf("%s\tSEMI COLON\n",yytext);}
","     {printf("%s\tCOMMA\n",yytext);}



{wrongId} {printf("\n\t%s identifier cant start with number\n",yytext);}
{identifier}  {printf("\n\t%s is a identifiers\n",yytext);}
"%d"|"%s"|"%c"|"%f"|"%e" {printf("%s\tFORMAT SPECIFIER\n",yytext); }
%%

int yywrap(void){}

int main()
{
    yylex();

    return 0;
}
```

Output:

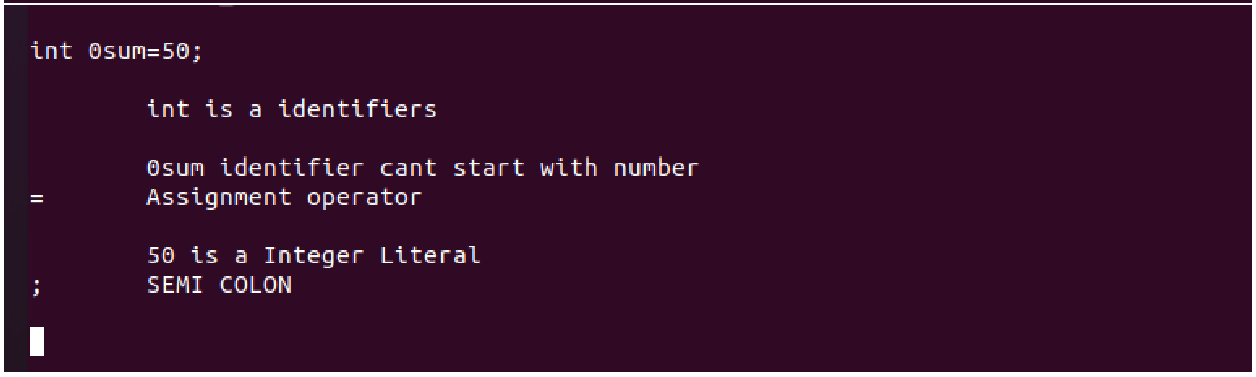


methul@methul: ~/compiler

```
methul@methul:~/compiler$ flex lexicalAnalyz.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){prntf("hello")}else{prntf("hi")}

        if is a identifiers
(       OPEN BRACKET

        k is a identifiers
<       GREATER THAN operator

        sum is a identifiers
)       CLOSE BRACKET
{       OPEN_CURLY BRACKET

        prntf 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

        prntf 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
```