

Ex No: 3

Date:

DEVELOP A LEXICAL ANALYZER TO RECOGNIZE TOKENS USING LEX TOOL

AIM:

To implement the program to identify C keywords, identifiers, operators, and statements like [], { } using LEX tool.

ALGORITHM:

- Configure lexer options with `%option noyywrap`.
- Define regular expressions for tokens like `letter`, `digit`, and `id`.
- Initialize a counter variable `n` to track line count.
- Define rules to identify language constructs such as keywords, function names, identifiers, numbers, operators, and preprocessor directives.
- Increment the line count for each newline character encountered.
- In the `main()` function, open the file "sample.c", perform lexical analysis with `yylex()`, and print the total number of lines processed.
-

PROGRAM:

```
%option noyywrap
letter [a-zA-Z]
digit [0-9]
id [_a-zA-Z]
AO [+|-|/|%|*]
RO [<|>|<=|>|=|==]
pp [#]
%{
int n=0;
%}

%%
"void"                printf("%s return type\n",yytext);
{letter}*[(|)]        printf("%s Function\n",yytext);
"int"|"float"|"if"|"else" printf("%s keywords\n",yytext);
"printf"              printf("%s keywords\n",yytext);
{id}({id}|{digit})*   printf("%s Identifier\n",yytext);
{digit}{digit}*       printf("%d Numbers\n",yytext);
{AO}                  printf("%s Arithmetic
Operators\n",yytext);
```

```
{RO}                                printf("%s Relational
Operators\n",yytext);
{pp}{letter}*[<]{letter}*[{.}{letter}]>] printf("%s processor
Directive\n",yytext);

[\n]                                n++;
".|\"'|\"{|\"";"                printf("%s others\n",yytext);
%%

int main()
{
    yyin=fopen("sample.c","r");
    yylex();
    printf("No of Lines %d\n",n);
}
```

OUTPUT:

```
[root@fedora student]# vi exp3_271.1
[root@fedora student]# lex exp3_271.1
[root@fedora student]# cc lex.yy.c
[root@fedora student]# ./a.out
#include<stdio.h> void main(){ int a,b; }
#include<stdio.h> processor Directive
void return type
main() Function
{ others
  int keywords|
  a Identifier
  , others
  b Identifier
; others
} others
```

RESULT: