COMPILER CONSTRUCTION

ASSIGNMENT-1

Submitted by
Ahamed Favas
CSE S6,No: 7

IMPLEMENTATION OF A LEXICAL ANALYZER USING LEX

AIM:

To write a program for implementing a Lexical analyser using LEX tool in Linux platform.

ALGORITHM:

Step1: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user_subroutines

Step2: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

Step3: In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

Step4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

Step5: When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

Step6: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.

Step7: The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then

receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

PROGRAM CODE:

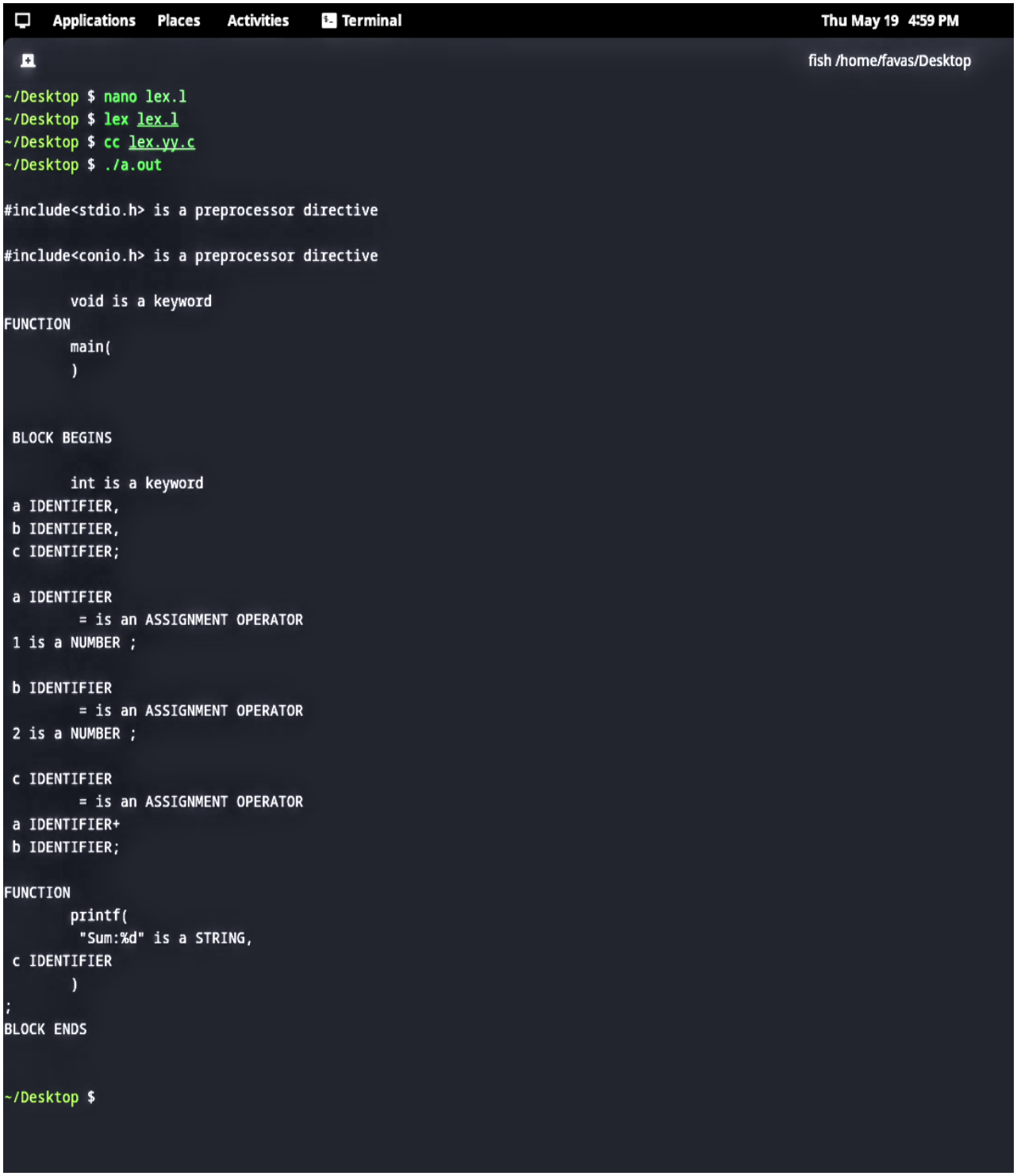
```
//Implementation of Lexical Analyzer using Lex tool
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{  {if(!COMMENT)printf("\n BLOCK BEGINS");}
\}  {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO;
```

```
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc, char **argv)
{
FILE *file;
file=fopen("var.c","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

INPUT:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

OUTPUT:


Applications Places Activities Terminal Thu May 19 4:59 PM
fish /home/favas/Desktop
~/Desktop $ nano lex.l
~/Desktop $ lex lex.l
~/Desktop $ cc lex.yy.c
~/Desktop $ ./a.out
#include<stdio.h> is a preprocessor directive
#include<conio.h> is a preprocessor directive
void is a keyword
FUNCTION
main(
)
BLOCK BEGINS
int is a keyword
a IDENTIFIER,
b IDENTIFIER,
c IDENTIFIER;
a IDENTIFIER
= is an ASSIGNMENT OPERATOR
1 is a NUMBER ;
b IDENTIFIER
= is an ASSIGNMENT OPERATOR
2 is a NUMBER ;
c IDENTIFIER
= is an ASSIGNMENT OPERATOR
a IDENTIFIER+
b IDENTIFIER;
FUNCTION
printf(
"Sum:%d" is a STRING,
c IDENTIFIER
)
;
BLOCK ENDS
~/Desktop $