National Institute of Technology Agartala

INDUSTRIAL TRAINING REPORT

Submitted by:

Name: Akash Debbarma

Enrollment No: 19UCS043

Section: B

Semester: 7

Department of Computer Science and Engineering

Objective :
Implement the lexical analyzer using JLex, flex or other lexical analyzer generating tools.

Resources:
Vs code, flex, GCC compiler

Program Logic:
Input : LEX specification files for the token

Output : Produces the source code for the Lexical Analyzer with the name lex.yy.c and displays the tokens from an input file.

1. Start
2. Open a file in text editor
3. Create a Lex specifications file to accept keywords, identifiers, constants, operators and relational operators in the following format. a) %{ Definition of constant /header files %} b) Regular Expressions %% Transition rules %% c) Auxiliary Procedure (main( ) function)
4. Save file with. l extension e.g. mylex.l
5. Call lex tool on the terminal e.g. [root@localhost]# flex filename.l. This lex tool will convert ".l" file into ".c" language code file i.e., lex.yy.c
6. Compile the file lex.yy.c using C compiler. e.g. gcc lex.yy.c. After compilation the file lex.yy.c, the output file is in a.out
7. Run the file a.out giving an input(text/file) e.g. ./a.out.
8. Upon processing, the sequence of tokens will be displayed as output.
9. Stop

Procedure:
run commands:
flex filename.l
cc lex.yy.cc -ll

./a.out

Program:

```
%{
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("./input.txt","r");
   if(!file)
   {
       printf("could not open the file");
       exit(0);
   }
   yyin=file;
   yylex();
   printf("\n");
   return(0);
}
int yywrap()
{
   return(1);
}
```

Input:

./input.txt

```
void sample(){
    int a = 10;
    int b = 10;
    return a + b;
}
```

Output:

```
 void is a keyword
FUNCTION
        sample(
        )

 BLOCK BEGINS

        int is a keyword
 a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 10 is a NUMBER ;

        int is a keyword
 b IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 10 is a NUMBER ;

        return is a keyword
 a IDENTIFIER +
 b IDENTIFIER;
BLOCK ENDS
```

Conclusion:

Thus in this experiment I learnt about implementing the lexical analyzer using flex lexical analyzer generating tools and run successfully.