SSC LAB ASSIGNMENT NO.5
SCANNER USING LEX TOOL

NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1

Title: Write a program using Lex specifications to implement lexical analysis phase of compiler to generate tokens of a subset of C program .

Aim: Generate lexical analyzer for Java/C language using LEX.

Objective:

1. To understand the lexical analysis phase of the compiler.
2. To understand the scanner for a subset of java.

Theory: Explain the following

1. Token lexeme and pattern.

Token

A token is a pain consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit.A particular denoting an identifier.The token names are the input symbol that are the parser process.
Typically tokens are ,
1) Identifiers
2) Keywords
3) Operator
4) special symbol 5)constant

Pattern

A pattern is a description of the form that the lexeme's of a token may take. In the case of keyword as a token , the pattern is just the sequence of characters that form the keyword.

For identifiers and some other tokens. The pattern is more complex structure.That is matched by many string.

Lexeme

A lexeme is a sequence of character in the source program that matches pattern for a token and identified by the lexical analyzer as an instance of that token.
Example : while(y > = t) = y -3

Will be represented by the set of pairs.

Lexeme	token
while	while
(	lparen
y	identifier
>=	Comparison
t	identifier
)	Rparen
y	identifier
=	Assignment
y	identifier
-	Arithmetic
3	integer
;	Finish of a statement

2. Use of regular expression (RE) in specifying lexical structure of a language.

The lexical analyzer needs to scan and identify only a finite set of valid string/token/lexeme that belong to the language in hand. It searches for the pattern defined by the language rules.

Regular expressions have the capability to express finite languages by defining a pattern for finite strings of symbols. The grammar defined by regular expressions is known as regular grammar. The language defined by regular grammar is known as regular language.

Regular expression is an important notation for specifying patterns. Each pattern matches a set of strings, so regular expressions serve as names for a set of strings. Programming language tokens can be described by regular languages. The specification of regular expressions is an example of a recursive definition. Regular languages are easy to understand and have efficient implementation.

If r and s are regular expressions denoting the languages L(r) and L(s), then

Union : (r)|(s) is a regular expression denoting L(r) U L(s)

Concatenation : (r)(s) is a regular expression denoting L(r)L(s)

Kleene closure : (r)* is a regular expression denoting (L(r))*

(r) is a regular expression denoting L(r)

Representing valid tokens of a language in regular expression

If x is a regular expression, then:

- x* means zero or more occurrence of x.
  i.e., it can generate { e, x, xx, xxx, xxxx, … }
- x+ means one or more occurrence of x.
  i.e., it can generate { x, xx, xxx, xxxx … } or x.x*
- x? means at most one occurrence of x
  i.e., it can generate either {x} or {e}.

  [a-z] is all lower-case alphabets of English language.
  [A-Z] is all upper-case alphabets of English language.
  [0-9] is all natural digits used in mathematics.

Representing occurrence of symbols using regular expressions

letter = [a – z] or [A – Z]

digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 or [0-9]

sign = [+ | -]

3. Format of lex specification file. (*.l).

```
1.  Definition Section:
%{      contains c code usually consisting of
        #define
        extern
        global declarations that you want to be made part of the
        code for y.tab.c
%}
optional macro definitions for regular expressions
%%
2. Rules section:

regular expression    { c code to be executed when this R.E. is matched against the source. }

%%
```

3. Code section:

- Usually lex is being used in conjunction with yacc. If so, the code section contains functions employed by your c code in the rules section.
- If the program constructed by lex is being used as a standalone program (and not being used in conjunctions with yacc), then the code section may optionally contain a main program.

To run the program on windows command prompt :

```
flex prog.l       (produces lex.yy.c)

gcc lex.yy.c      (produces a --- the executable file)

a.exe             (to run the program)
```

Input: Subset of C language.

```
#include<stdio.h>
void main()
{
int b[6];
string s;
s="a1n2u";
int c,d;
b[0]=20.54;
c=10;
d=4+c;
if(d>10)
{
printf("Greator than 10");
}
/*
inside a multiline comment
*/
//inside a singleline comment
printf("The sum is");
getch();
}
```

Output: Sequence of tokens generated by lexical analyzer and Symbol Table.

Platform: Windows Command Prompt and lex tool.

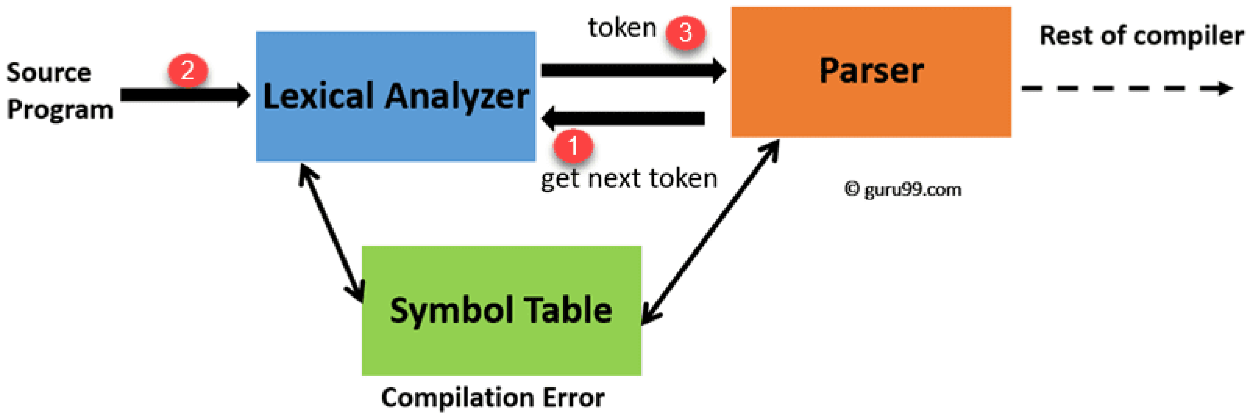
Conclusion: Implemented scanner in C.

FAQs:

1. **Give various tasks performed during lexical analysis phase.**

The main task of lexical analysis is to read input characters in the code and produce tokens.

Lexical analyzer scans the entire source code of the program. It identifies each token one by one. Scanners are usually implemented to produce tokens only when requested by a parser. Here is how this works-



1. "Get next token" is a command which is sent from the parser to the lexical analyzer.
2. On receiving this command, the lexical analyzer scans the input until it finds the next token.
3. It returns the token to Parser.

Lexical Analyzer skips whitespaces and comments while creating these tokens. If any error is present, then Lexical analyzer will correlate that error with the source file and line number.

Lexical analyzer performs below given tasks:

- Helps to identify token into the symbol table
- Removes white spaces and comments from the source program

- Correlates error messages with the source program
- Helps you to expands the macros if it is found in the source program
- Read input characters from the source program

2. What is the role of RE, DFA in lexical analysis?

The collection of tokens of a programming language can be specified by a set of regular expressions. A scanner or lexical analyzer for the language uses a DFA (recognizer of regular languages) in its core. Different final states of the DFA identifies different tokens. Synthesis of this DFA from the set of regular expressions can be automated.

3. What is LEX?

Lex is a program that generates lexical analyzer. It is used with YACC parser generator.The lexical analyzer is a program that transforms an input stream into a sequence of tokens.It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

- ***lex.l*** is an a input file written in a language which describes the generation of lexical analyzer. The lex compiler transforms *lex.l* to a C program known as *lex.yy.c.*
- ***lex.yy.c*** is compiled by the C compiler to a file called *a.out.*

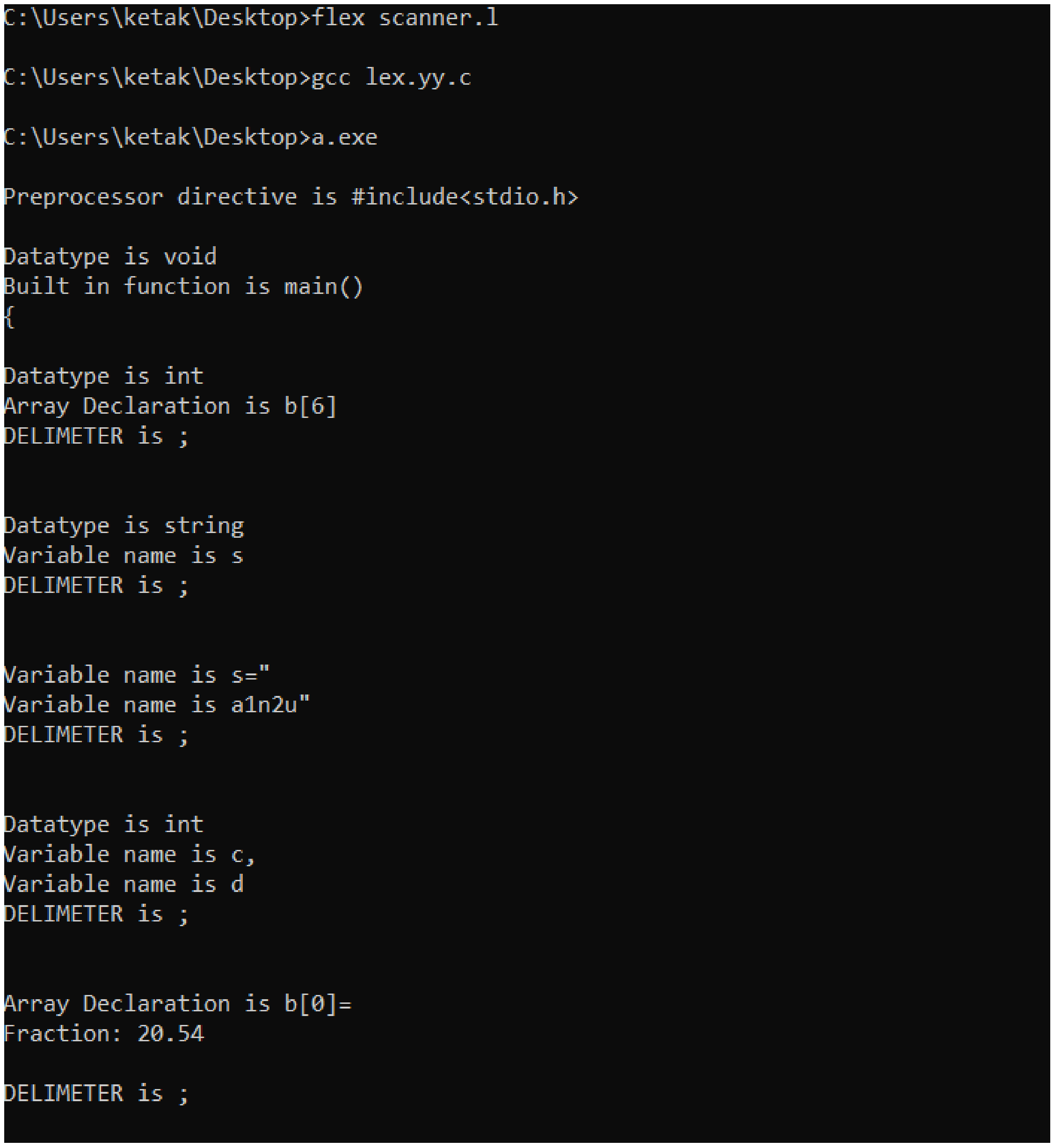
PROGRAM :

```
/*
 SSC LAB ASSIGNMENT NO.5
SCANNER USING LEX TOOL
NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1
 */

%option noyywrap
%{
   #include<stdio.h>
   #include<string.h>
   struct SymbolTable
   {
   char symbol[10];
   char type[10];
   }SymbolTable[10];
```

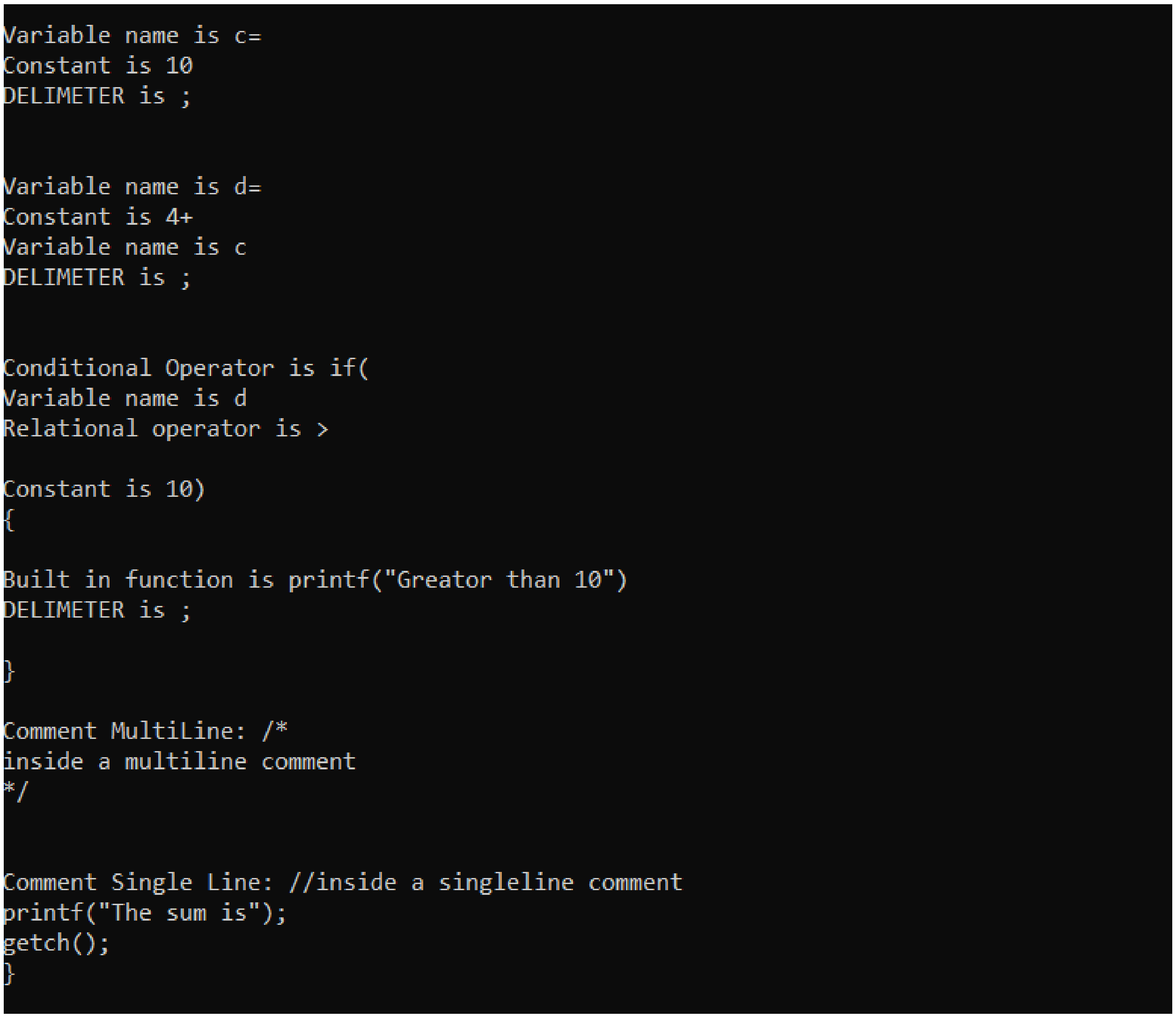
```
int count=0;
char data[10];
char type[10];
void insert();
void display();
%}
ws[\t]
letter [a-zA-Z]
digit [0-9]
id({letter}({letter}|{digit})*)
notid [0-9]*[a-zA-Z0-9]*
Commentmulti [/]{1}[*]{1}[^*]*[*]{1}[/]
Commentsingle [/]{1}[/]{1}[^*]*
FRACTION [0-9]*{1}[.]{1}[0-9]*
datatype(void|int|float|char|string)
keywords(main|getch|clrscr)
keyword1(if|else|case|while|do|for|switch{ws}*\(.*\))
%%
#include{ws}*<{id}\.h>   {printf("\nPreprocessor directive is
%s\t",yytext);}
{FRACTION} {printf("\nFraction: %s\n",yytext);}
{digit}+           { printf("\nConstant is %s",yytext);}
{datatype}         { printf("\nDatatype is %s",yytext);}
printf{ws}*\(\".*\"\) |
scanf{ws}*\(\".*\"\)    { printf("\nBuilt in function is %s",yytext);}
{keyword1}/"("          { printf("\nConditional Operator is %s",yytext);}
{keywords}/"("     { printf("\nBuilt in function is %s",yytext);}
{id}/"("           {printf("\nUser defined function is
%s",yytext);insert(yytext,"function");}
{id}"["{digit}*"]" { printf("\nArray Declaration is
%s",yytext);insert(yytext,"array declaration");}
{id}               { printf("\nVariable name is
%s",yytext);insert(yytext,"id");}
"<"|"<="|">"|">+"|"=="|"!=" {printf("\nRelational operator is
%s\n",yytext); }
{notid} {printf("\nInvalid Identifier is %s\n",yytext);}
{Commentmulti}          {printf("\nComment MultiLine: %s\n",yytext);}
{Commentsingle}    {printf("\nComment Single Line: %s\n",yytext);}
(\;)               {printf("\nDELIMETER is %s\n",yytext);}

%%

int main()
{
   yyin = fopen("input.txt","r");
   yylex();
   display();
   return 0;
}

void insert(char data[10],char type[10])
{
   strcpy(SymbolTable[count].symbol,data);
   strcpy(SymbolTable[count].type,type);
   ++count;
}
```

```
void display()
{
   int i;
   for(i=0;i<count;i++)
   {
         printf("\n%s\t%s",SymbolTable[i].symbol,SymbolTable[i].type);
   }
}
```

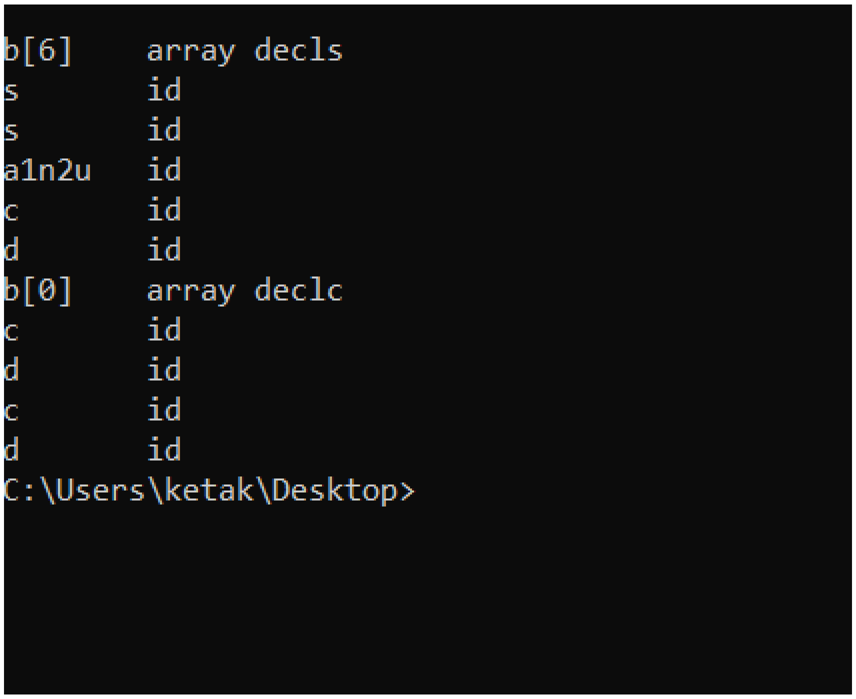
OUTPUT SS :



```
C:\Users\ketak\Desktop>flex scanner.l

C:\Users\ketak\Desktop>gcc lex.yy.c

C:\Users\ketak\Desktop>a.exe

Preprocessor directive is #include<stdio.h>

Datatype is void
Built in function is main()
{

Datatype is int
Array Declaration is b[6]
DELIMETER is ;


Datatype is string
Variable name is s
DELIMETER is ;


Variable name is s="
Variable name is a1n2u"
DELIMETER is ;


Datatype is int
Variable name is c,
Variable name is d
DELIMETER is ;


Array Declaration is b[0]=
Fraction: 20.54

DELIMETER is ;
```



```
Variable name is c=
Constant is 10
DELIMETER is ;


Variable name is d=
Constant is 4+
Variable name is c
DELIMETER is ;


Conditional Operator is if(
Variable name is d
Relational operator is >

Constant is 10)
{

Built in function is printf("Greator than 10")
DELIMETER is ;

}

Comment MultiLine: /*
inside a multiline comment
*/


Comment Single Line: //inside a singleline comment
printf("The sum is");
getch();
}
```



```
b[6]    array decls
s       id
s       id
a1n2u   id
c       id
d       id
b[0]    array declc
c       id
d       id
c       id
d       id
C:\Users\ketak\Desktop>
```