SSGMCE	SHRI SANT GAJANAN MAHARAJ COLLEGE OF ENGG.	LABORATORY MANUAL
	PRACTICAL EXPERIMENT INSTRUCTION SHEET	
	EXPERIMENT TITLE: Write a LEX Program to scan reserved words and Identifiers of C Language.	
EXPERIMENT NO.: SSGMCE/WI/IT/01/6IT01/06	ISSUE NO.: 00	ISSUE DATE: 01.02.2022
REV. DATE:	REV. NO.:	DEPTT.: INFORMATION TECHNOLOGY
LABORATORY: COMPILER DESIGN (CD)	SEMESTER: VI	PAGE: OF 6

1.0) AIM:

Write a LEX Program to scan reserved words and Identifiers of C Language.

2.0) OBJECTIVE:

After the completion of this experiment, LEX Program will be able to scan reserved words and Identifiers of C Language.

3.0) FACILITIES/ APPARATUS:

i. Hardware: Computer Machine

ii. Software: FLEX (fast lexical analyzer generator) for Windows- LEX and YACC (Bison) Installer for Windows 7/8.1/10 32-bit & 64-bit

4.0) THEORY:

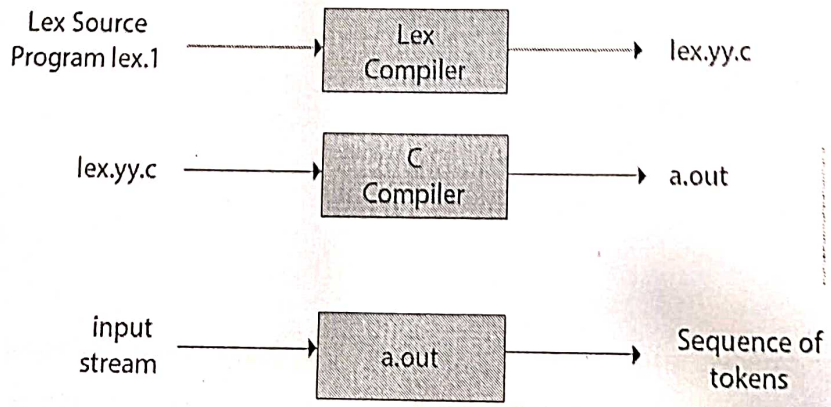
LEX is a program that generates a lexical analyzer. It is used with the YACC parser generator. The lexical analyzer is a program that transforms an input stream into a sequence of tokens.It reads the input stream and produces the source code as output by implementing the lexical analyzer in the C program.

LEX Functions:

1) Firstly, the lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.

2) Finally, the C compiler runs the lex.yy.c program and produces an object program a.out.

3) a.out is a lexical analyzer that transforms an input stream into a sequence of tokens.



LEX Program Structure:

In the input file, there are 3 sections:

1. Definition Section: The definition section contains the declaration of variables, regular definitions, and manifest constants. In the definition section, the text is enclosed in **"%{%}"** brackets. Anything written in these brackets is copied directly to the file **lex.yy.c**

Syntax:

```
%{
        // Definitions
%}
```

2. Rules Section: The rules section contains a series of rules in the form: *pattern action* and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in **"%% %%"**.

Syntax:

```
%%
        pattern action
%%
```

3. User Code Section: This section contains C statements and additional functions. We can also compile these functions separately and load them with the lexical analyzer.

Basic Program Structure:

```
%{
        // Definitions
%}
%%
        Rules
%%
User code section
```

FLEX (fast lexical analyzer generator) is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with the Berkeley

YACC parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than LEX and YACC and produce faster code.

Bison produces a parser from the input file provided by the user. The function **yylex()** is automatically generated by the flex when it is provided with a **.l file** and this yylex() function is expected by the parser to call to retrieve tokens from the current/this token stream.

Note: The function yylex() is the main flex function that runs the Rule Section and extension (.l) is the extension used to save the programs.

Contents / Salient Features of Flex Windows

1) In-built GCC/G++/cc Libraries of Linux: The Flex Windows Package contains inbuilt GCCand g++ libraries [c and c++ compilers] which are ported to windows officially by MinGW and are actively developed by the Linux Open-Source Community.

2) LEX and YACC Package Binaries: The package contains the latest updated versions of LEX and YACC binaries [flex and bison] which are developed by their developers. The original binaries are included as-it-is in the package so as to ensure smooth and error-free compilation and build of programs.

3) Pre-Configured EditPlus IDE: The package also contains EditPlus IDE which contains pre-defined Blank templates for the LEX/YACC/C/C++/Java Files, thus each time userwants to type a program to simply use the New LEX / New YACC template, and the basic code will be inserted thus saving time and efforts.

4) The EditPlus IDE also contains user Commands for **LEX Compile, YACC Compile,LEX Build, LEX+YACC Build, and Band for Execute.** thus, saving time to type complete commands like "lex abc.l" or cc lex.yy.c.

Method to Run Programs through CMD

i) Click on Execute CMD directly button in the IDE.

ii) Compile the Lex File by typing the command lex <filename>.l

iii) Build the LEX File by gcc/cc command in the CMD e.ggcclex.yy.c -o <executable name for program

iv) Execute the program by typing <executable name for the program>.exe

v) **The -o <executable name for program>**parameter is optional, you can skip the said parameter by directly building by GCClex.yy.cand then directly executing program by typing a.exe

5.0) PROGRAM:

```
%{
int COMMENT=0;
%}
identifier[a-z|A-Z][a-z|A-Z|0-9]*

%%
#.* {printf("\n %s is a preprocesor directive",yytext);}
int {printf("\n\t %s is a keyword",yytext);}
float {printf("\n\t %s is a keyword",yytext);}
double {printf("\n\t %s is a keyword",yytext);}
char {printf("\n\t %s is a keyword",yytext);}
if {printf("\n\t %s is a keyword",yytext);}
else {printf("\n\t %s is a keyword",yytext);}
while {printf("\n\t %s is a keyword",yytext);}
do {printf("\n\t %s is a keyword",yytext);}
return {printf("\n\t %s is a keyword",yytext);}
break {printf("\n\t %s is a keyword",yytext);}
continue {printf("\n\t %s is a keyword",yytext);}
void {printf("\n\t %s is a keyword",yytext);}
switch {printf("\n\t %s is keyword",yytext);}
for {printf("\n\t %s is a keyword",yytext);}
typedef {printf("\n\t %s is a keyword",yytext);}
struct {printf("\n\t %s is a keyword",yytext);}
goto {printf("\n\t %s is a keyword",yytext);}
"/*" {COMMENT=1;}
"*/" {COMMENT=0;}
{identifier}\( {if(!COMMENT) printf("\n FUNCTIONS \n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n BLOCK ENDS");}
```

```
{identifier} {if(!COMMENT) printf("\n %s is an IDENTIFIER",yytext);}
{identifier}(\[[0-9]*\])?\( {if(!COMMENT)
printf("\n %s is an IDENTIFIER",yytext);}
\".*\" {if(COMMENT)printf("\n\t %s is a string",yytext);}
[0-9]+ {if(COMMENT)printf("\n\t %s is a number",yytext);}
\)(\;)? {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);}
\\n
%%

int main(int argc,char **argv)
{
        if(argc>1)
        {
                FILE *file;
                file=fopen(argv[1],"r");
                if(!file)
                {
                        printf("COULD NOT OPEN %s \n",argv[1]);
                        exit(1);
                                }
                yyin=file;
        }
        yylex();
        printf("\n\n");
return 0;
}

int yywrap()
```

```
{
        return 0;
}
```

6.0) OUTPUT OF PROGRAM

INPUT

sum

OUTPUT

sum is an IDENTIFIER

INPUT

int

OUTPUT

int is a keyword

INPUT

Sagar123

OUTPUT

Sagar123 is a keyword

7.0) CONCLUSION:

A lexical analyzer has been designed using LEX Program to scan reserved words and Identifiers of C Language.