Practical No. 6

Aim: Write a program to generate Lexical Analyzer using FLEX tool.

Requirement: FLEX Tool, GCC

Theory:

Steps to download and install Flex:

a. On Windows

1. Download Flex Windows setup file for Windows 10 from: https://techapple.net/2014/07/flex-windows-lex-and-yacc-flex-and-bison-installer-for-windows-xp788-1/
2. Follow the steps and install the “Flex Windows”.
3. Open the folder at location “C:\Flex Windows\Lex\bin”. In this “bin” folder, there are two exe files named as “flex.exe” and “flex++.exe”.
4. Flex is for generating C program while flex++ is for generating C++ program as a lexical analyzer.
5. You can either write down the complete lex specification code as shown below, using “Flex.exe” command prompt, or can save the specification file with extension “.l”.

b. On Linux Ubuntu

1. Update the packages first:
```
sudo apt-get update
```
2. Install flex:
```
sudo apt-get install flex
```
3. If ‘dpkg’ erro appears, run:
```
sudo dpkg --configure -a
```
   Run again:
```
sudo apt-get install flex
```
4. To uninstall:
```
sudo apt-get purge flex
```

Structure of a LEX Code:

Following is a structure of a flex code called LEX specification file.

Definition section
%%
Rules section
%%
C code section

Where;
The **definition** section is the place to define macros and to import header files written in C. It is also possible to write any C code here, which will be copied verbatim into the generated source file.
The **rules** section is the most important section; it associates patterns with C statements. Patterns are simply regular expressions. When the lexer sees some text in the input matching a given pattern, it executes the associated C code. This is the basis of how lex operates.
The **C code** section contains C statements and functions that are copied verbatim to the generated source file. These statements presumably contain code called by the rules in the rules section. In large programs it is more convenient to place this code in a separate file and link it in at compile time.

Program:

```
/*** Definition section ***/
%{
/* C code to be copied verbatim */
#include<stdio.h>
%}

/* [0-9]+ matches a string of one or more digits */
DIGIT           [0-9]+
ID              [a-z][A-Za-z0-9]*
%option noyywrap

/* This tells flex to read only one input file */
%option noyywrap

%%
/*** Rules section ***/

{DIGIT}+        {
                /* yytext is a string containing the matched text. */
                printf( "An integer: %s\n", yytext);
                }

{DIGIT}+"."{DIGIT}*     {
                        printf( "A float: %s\n", yytext);
                        }

if|else|for|do|while    {
                        printf( "A keyword: %s\n", yytext);
                        }
```

```
{ID}                    {
                        printf( "An identifier: %s\n", yytext);
                        }

"+"|"-"|"*"|"/"         {
                        printf( "An operator: %s\n", yytext);
                        }

"{"[^}\n]*"}"           /* eat up one-line comments */

[ \t\n]+                /* eat up whitespace */

.                       {
                        printf( "Unrecognized character: %s\n", yytext);
                        }
%%
/*** C Code section ***/
int main()
{
        yylex();   /* Call the lexer, then quit. */
        return 0;
}
```

Saving lex specification file on Linux and executing it:
The above code is saved with an extension '.l'. Using a command '**flex program_name.l**' on a terminal, a file called '**lex.yy.c**' is automatically created at the same location where lex code is saved. This is the c program containing lexical analyzer. We can change the name of **lex.yy.c** to any other name such as **P1.c**.Compile this lex.yy.c using any C compiler such as gcc with a command '**gcc lex.yy.c**' and then execute it using a command **'./a.out**':

Output:

//Paste a color printout of the output here.

Conclusion:

Thus, a lexical analyzer is implemented successfully using a lex generator tool Flex.