Ex. No. 1.b

Date:

**Implementation of lexical analyzer using LEX TOOL**

**AIM**

To Implementation of lexical analyzer using LEX TOOL.

**ALGORITHM**

**Step 1:** **Write the Lex File**

* Write the Lex (Flex) file with pattern matching rules and actions. The file typically has a .l extension (e.g., Pro1.l).

**Step 2:** **Install Required Tools**

* Ensure that Flex (for generating lexical analyzers) and GCC (GNU Compiler Collection) are installed on your system.

**Step 3:** **Set Environment Variables**

* Add the directories where Flex (flex.exe) and GCC (gcc.exe) are installed to your system's PATH environment variable so that they can be accessed from the command line.

**Step 4:** **Generate C Source Code Using Flex**

* Open the command prompt or terminal.
* Navigate to the directory containing the Lex file.

Run the Flex tool to generate the C source code:  
flex Pro1.l

* This will generate a file named lex.yy.c.

**Step 5:** **Compile the Generated C File**

Use GCC to compile the generated C file (lex.yy.c) into an executable:  
gcc lex.yy.c -o output.exe

* This will create an executable file named output.exe (or any name you choose).

**Step 6:** **Run the Executable**

Run the generated executable to process the input according to the patterns defined in the Lex file:  
./output.exe

* The program will read input (either from standard input or a file, depending on how the lexer is set up) and print the corresponding actions defined in the Lex file.

**Step 7:** **Interpret the Output**

* Review the output generated by the program, which will include messages or actions based on the input provided and the rules defined in the Lex file.

**Step 8:** **Debug and Modify (if necessary)**

* If there are errors or unexpected behavior, debug the Lex file, re-run Flex and GCC, and test the executable again.

**PROGRAM**

%{

%}

identifier[a-zA-Z][a-zA-Z0-9]\*

%%

#.\* printf("\n%s is PREPROCESSOR DIRECTIVE\n",yytext);

int |

float |

double |

char |

for |

if printf("%s is a keyword\n",yytext);

{identifier}\( printf("\n\n FUNCTION CALL\n %s",yytext);

\{

printf("BLOCK BEGINS\n");

\}

printf("BLOCK ENDS\n");

= printf("%s is a ASSIGNMENT OPERATOR\n",yytext);

[0-9]+ printf("%s is NUMBER\n",yytext);

\< |

\> |

\== |

\>= |

\<= printf("%s is a RELATIONAL OPERATOR\n",yytext);

\( { ECHO;printf("\n");}

\) { ECHO; printf("\n");}

\+ |

\- |

\\* printf("%s is a ARITHMETIC OPERATOR \n");

\++ printf("%s is a INCREMENTAL OPERATOR\n");

\; { ECHO; printf("\n");}

%%

main()

{

yylex();

}

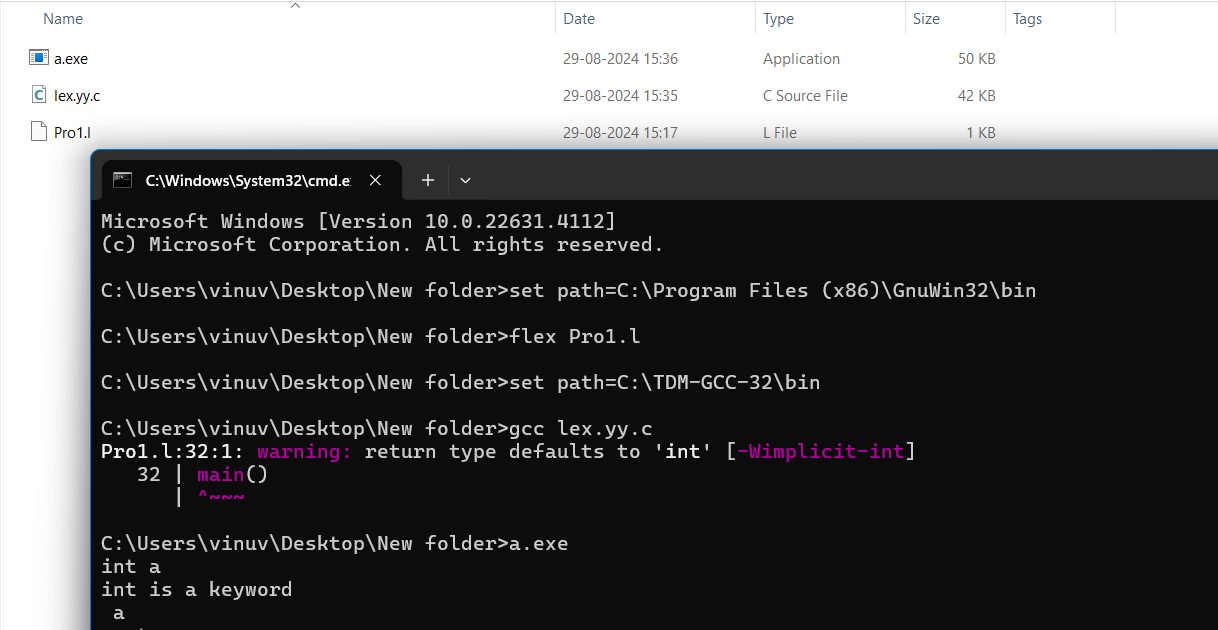
int yywrap()

{

return 1;

}

**OUTPUT**

****

**RESULT**

Thus to Implementation of lexical analyzer using LEX TOOL has been executed and verified successfully.