|  |  |
| --- | --- |
| **Ex No: 6** | **Token Separation using LEX Tool** |
| **Date of Exercise** | **10/02/2022** |

|  |
| --- |
| **AIM** |
| To write a lex specification for performing token separation for a given input source program. |
| **DESCRIPTION** |
| Lex helps write programs whose control flow is directed by instances of regular  expressions in the input stream. It is well suited for editor-script type transformations and for  segmenting input in preparation for a parsing routine. Lex source is a table of regular  expressions and corresponding program fragments. The table is translated to a program which  reads an input stream, copying it to an output stream and partitioning the input into strings  which match the given expressions. As each such string is recognized the corresponding  program fragment is executed. The recognition of the expressions is performed by a  deterministic finite automaton generated by Lex. The program fragments written by the user  are executed in the order in which the corresponding regular expressions occur in the input  stream. Lex turns the user&#39;s expressions and actions (called source in this memo) into the host  general-purpose language; the generated program is named yylex. The yylex program will  recognize expressions in a stream (called input in this memo) and perform the specified  actions for each expression as it is detected. The general format of Lex source is:  {definitions} %% {rules} %% {user subroutines} |
| **ALGORITHM** |
| 1. Create a lex specification file pgm.l 2. Include the required header files if necessary. 3. Declare the necessary variables, constants and regular definition. 4. Write the regular expressions and the corresponding actions for identifier, keyword, numbers, operators and punctuation. 5. Write the main function and call yylex() |
| **SAMPLE INPUT & OUTPUT** |
| **Input:**  int a,b,c; c=a+b\*20  **Output:**   &lt;keyword, int&gt; &lt;id1, a&gt;  &lt;punctuation, ,&gt; &lt;id2, b&gt;  &lt;punctuation, ,&gt; &lt;id3, c&gt;  &lt;punctuation, ;&gt; &lt;id3,c&gt;  &lt;op, =&gt; &lt;id1, a&gt;  &lt;op, +&gt; &lt;id2,b&gt;  &lt;op, \*&gt;  &lt;num, 20&gt;  &lt;punctuation, ;&gt; |
| **Code:** |

%{

#include<stdio.h>

int id=0;

int totalToken = 0;

%}

%%

asm|double|new|switch|auto|else|operator|template|break|enum|private|this|case|extern|protected|throw|catch|float|public|try|char|for|register|typedef|class|friend|return|union|const|goto|short|unsigned|continue|if|signed|virtual|default|inline|sizedof|void|delete|int|static|volatile|do|long|struct|while {printf("<KEYWORD, "); ECHO; printf(" >"); totalToken++;}

[{};,()] {printf("\n<Punctuation, "); ECHO; printf(" >"); totalToken++;}

[+-/=%] {printf("\n<OPERATOR, "); ECHO; printf(" >"); totalToken++;}

"END" {return -1;};

([a-zA-Z][0-9])+|[a-zA-Z] {printf("\n<Identifier, "); ECHO; printf(" id=%d >",id+1); id++; totalToken++;}

.|\n ;

%%

int yywrap()

{

return 1;

}

int main(void)

{

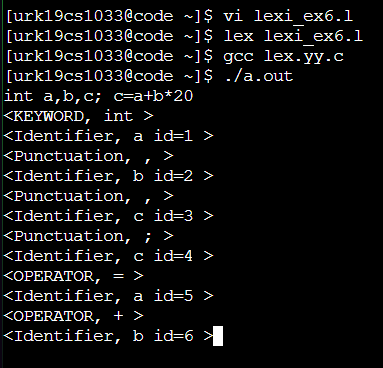
yylex();

printf("Total no of token in the program: %d",totalToken+1);

return 0;

}

**OUTPUT SCREENSHOT:**



**RESULT:**

Thus the program ran successfully and the output was verified.