PROBLEM DESCRIPTION:

Step1: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user_subroutines.

Step2: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

Step3: In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

Step4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

Step5: When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

Step6: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.

Step7: The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

Input:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

Output:

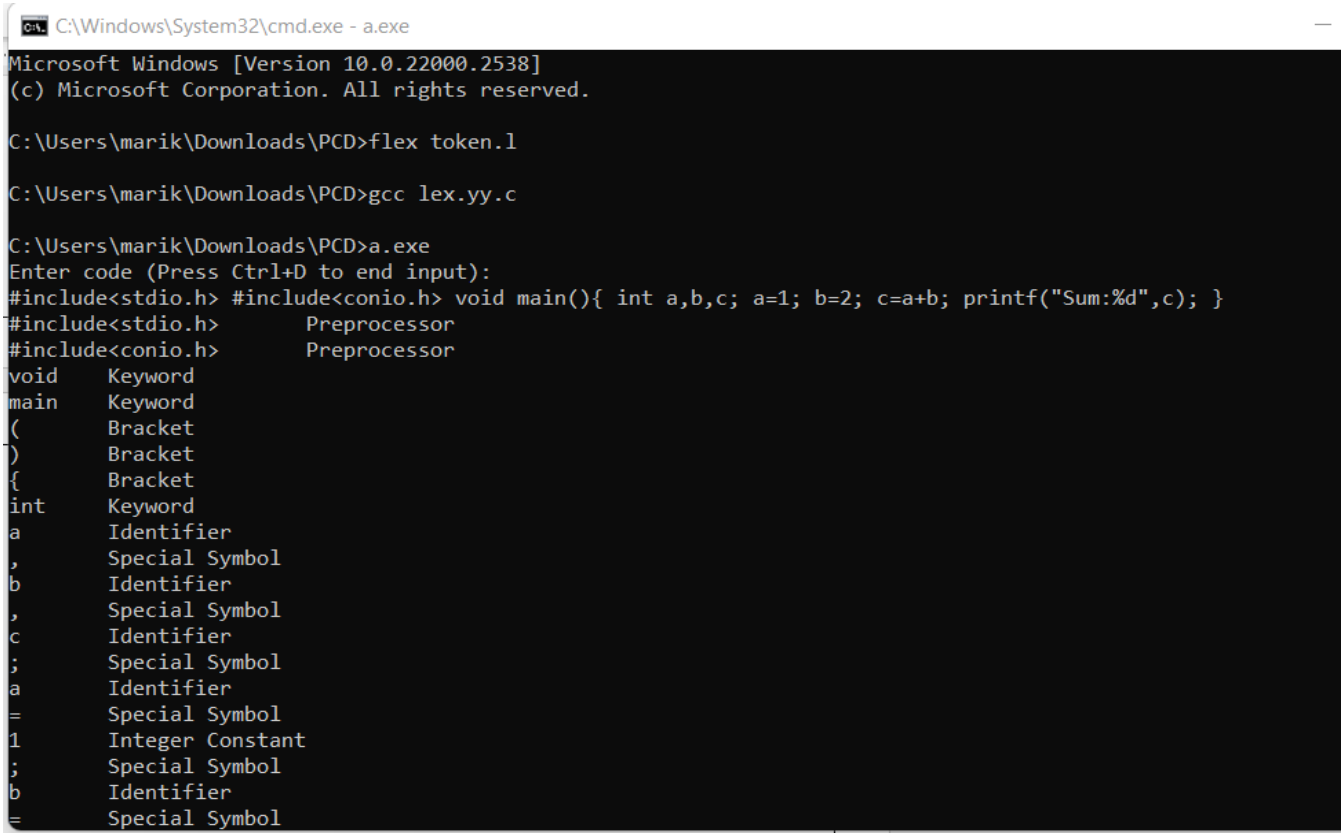
```
#include<stdio.h> is a preprocessor directive
#include<conio.h> is a preprocessor directive
void is a Keyword
main is a Keyword
( is a Symbol
) is a Symbol
{  is a Symbol
```

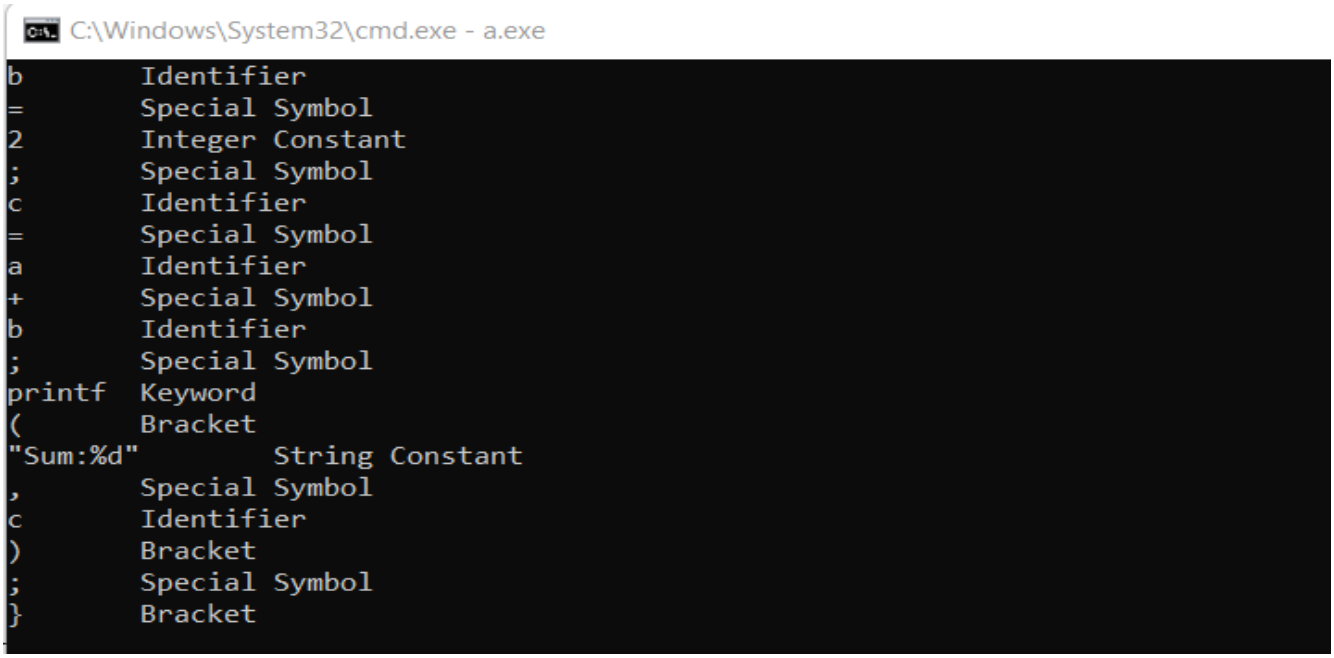
PROGRAM:

```
%{

#include <stdio.h>
#include <string.h>
#include <ctype.h>
void classify_token(char *token);

%}

%%

[ \t\n]+
\"(\\.|[^"\\])*\"        { printf("%s\tString Constant\n", yytext); }
#[a-zA-Z_]+              { printf("%s\tPreprocessor\n", yytext); }
"#include"("<"[a-zA-Z_]+".h">)  { printf("%s\tPreprocessor\n", yytext); }
[0-9]+\.[0-9]+           { printf("%s\tFloat Constant\n", yytext); }
[0-9]+                   { printf("%s\tInteger Constant\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { classify_token(yytext); }
```

```
"=="|"!="|"<="|">="|"+="|"-="|"*="|"/="|"&&"|"||" { printf("%s\tOperator\n", yytext); }

[-+*/=%&|^!<>?:;,[\]] { printf("%s\tSpecial Symbol\n", yytext); }

[\(\)\{\}\[\]]        { printf("%s\tBracket\n", yytext); }

.                     { printf("%s\tUnclassified Token\n", yytext); }

%%

void classify_token(char *token) {
   char *keywords[] = {"auto", "break", "case", "char", "const", "continue", "default", "do",
                       "double", "else", "if", "int", "long", "public", "register", "return", "short",
                       "sizeof", "struct", "switch", "typedef", "union", "unsigned", "void", "while",
                       "printf", "scanf", "main", NULL};

    for (int i = 0; keywords[i] != NULL; i++) {
       if (strcmp(token, keywords[i]) == 0) {
           printf("%s\tKeyword\n", token);
          return;
       }
   }

   printf("%s\tIdentifier\n", token); // If it is not a keyword, classify it as an identifier.
}

int yywrap() {
   return 1;
}

int main() {
   printf("Enter code (Press Ctrl+D to end input):\n");
   yylex();
   return 0;
}
```

OUTPUT


C:\Windows\System32\cmd.exe - a.exe
Microsoft Windows [Version 10.0.22000.2538]
(c) Microsoft Corporation. All rights reserved.
C:\Users\marik\Downloads\PCD>flex token.l
C:\Users\marik\Downloads\PCD>gcc lex.yy.c
C:\Users\marik\Downloads\PCD>a.exe
Enter code (Press Ctrl+D to end input):
#include<stdio.h> #include<conio.h> void main(){ int a,b,c; a=1; b=2; c=a+b; printf("Sum:%d",c); }
#include<stdio.h> Preprocessor
#include<conio.h> Preprocessor
void Keyword
main Keyword
(Bracket
) Bracket
{ Bracket
int Keyword
a Identifier
, Special Symbol
b Identifier
, Special Symbol
c Identifier
; Special Symbol
a Identifier
= Special Symbol
1 Integer Constant
; Special Symbol
b Identifier
= Special Symbol


C:\Windows\System32\cmd.exe - a.exe
b Identifier
= Special Symbol
2 Integer Constant
; Special Symbol
c Identifier
= Special Symbol
a Identifier
+ Special Symbol
b Identifier
; Special Symbol
printf Keyword
(Bracket
"Sum:%d" String Constant
, Special Symbol
c Identifier
) Bracket
; Special Symbol
} Bracket

Program understanding & Algorithm (10)	Implementation (30)	Viva (10)	Total (50)

Result:

Thus the program was executed using the lex tool and the output was verified.