RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	19CSC305A
Course Name	Compilers
Programme	B. Tech
Department	Computer Science and Engineering
Faculty	Engineering and Technology

Name of the Student	Deepak R
Reg. No	18ETCS002041
Semester/Year	5th/2020
Course Leader/s	Mr. Hari Krishna S.

Declaration Sheet			
Student Name	Deepak R		
Reg. No	18ETCS002041		
Programme	B. Tech	Semester/Year	5th/2020
Course Code	19CSC305A		
Course Title	Compilers		
Course Date		to	
Course Leader	Mr. Hari Krishna S		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	05/12/2020
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	**Programme**	B. Tech in Computer Science and Engineering
Semester/Batch	05th /2018		
Course Code	19CSC305A	**Course Title**	Compilers
Course Leader	Mr. Hari Krishna S. M. & Ms. Suvidha		

Assignment			
Register No.		**Name of the Stud ent**	

Sections		Marking Scheme	Marks		
			Max Marks	First Marks	Examiner Moderator
Part A 1					
	A 1.1	Implementation in *Lex*	06		
	A 1.2	Results and Comments	04		
		Part-A 1 Max Marks	10		
	A 2.1	Implementation in *Lex*	10		
	A 2.2	Results and Comments	05		
		Part-A 2 Max Marks	15		
		Total Assignment Marks	25		

Course Marks Tabulation

Component- CET B Assignment	First Examiner	Remarks	Second Examiner	Remarks
A.1				
A.2				
Marks (out of 25)				

Signature of First Examiner Signature of Moderator

Please note:

1. **Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.**
2. **The First Examiner is required to mark the comments in RED ink and the Second Examiner's comments should be in GREEN ink.**
3. **The marks for all the questions of the assignment have to be written only in the Component – CET B: Assignment table.**
4. **If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.**

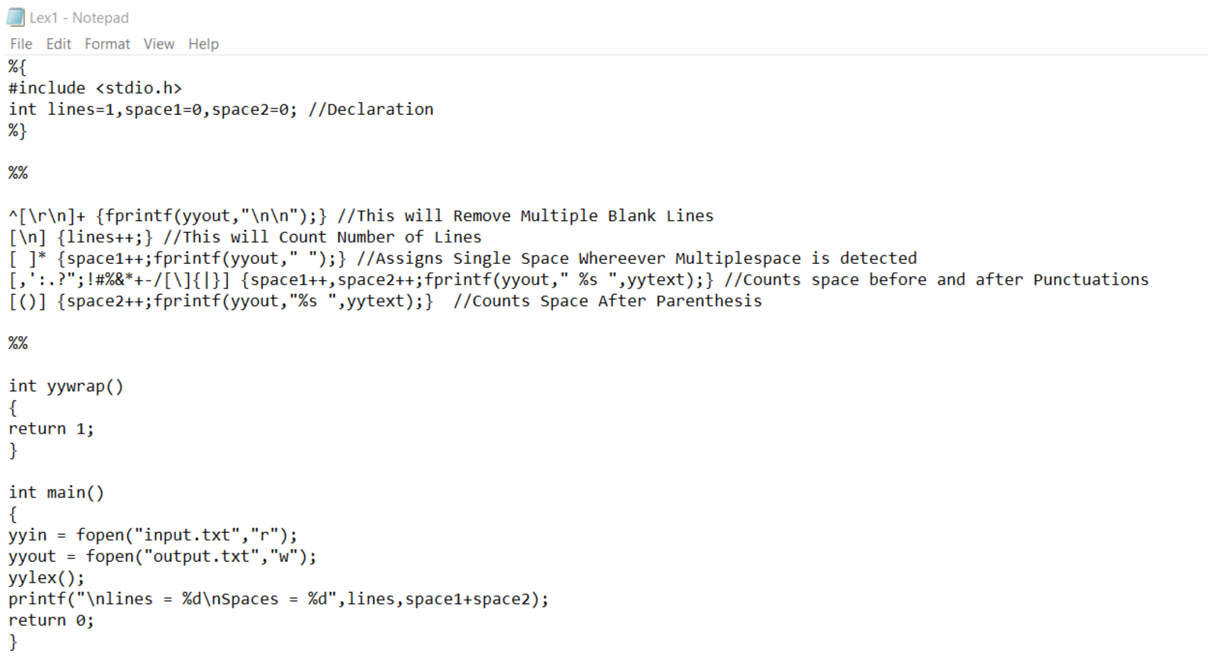
Assignment

Instructions to students:

1. **The assignment consists of 1 questions: Part A – 2 Question.**
2. **Maximum marks is 25.**
3. **The assignment has to be neatly word processed as per the prescribed format.**
4. **The maximum number of pages should be restricted to 25.**
5. **The printed assignment must be submitted to the course leader.**
6. **Submission Date: 28th November 2020**
7. **Submission after the due date is not permitted.**
8. **IMPORTANT: It is essential that all the sources used in preparation of the assignment must be suitably referenced in the text.**

Solution for A.1

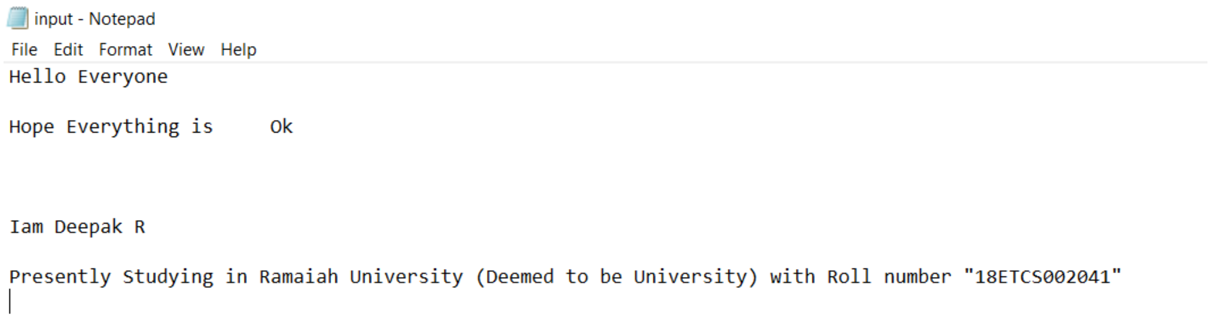
Lex1.1



```
%{
#include <stdio.h>
int lines=1,space1=0,space2=0; //Declaration
%}

%%

^[\r\n]+ {fprintf(yyout,"\n\n");} //This will Remove Multiple Blank Lines
[\n] {lines++;} //This will Count Number of Lines
[ ]* {space1++;fprintf(yyout," ");} //Assigns Single Space Whereever Multiplespace is detected
[,':.?";!#%&*+-/[\]{|}] {space1++,space2++;fprintf(yyout," %s ",yytext);} //Counts space before and after Punctuations
[()] {space2++;fprintf(yyout,"%s ",yytext);}  //Counts Space After Parenthesis

%%

int yywrap()
{
return 1;
}

int main()
{
yyin = fopen("input.txt","r");
yyout = fopen("output.txt","w");
yylex();
printf("\nlines = %d\nSpaces = %d",lines,space1+space2);
return 0;
}
```

Fig2.1 Lex Program to Compress Multiple lines ,spaces and to give space when Punctuations or Brackets are identified

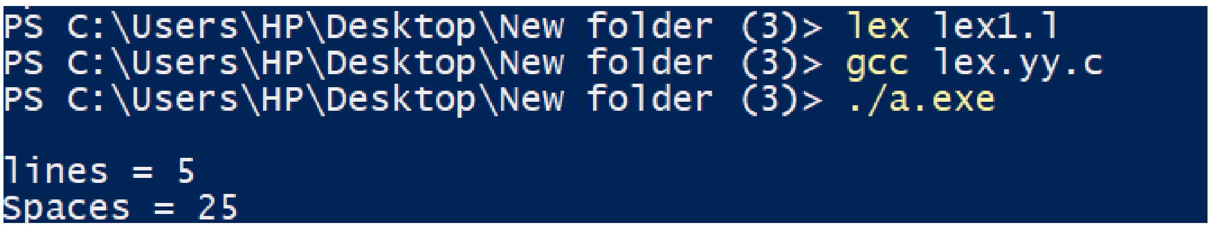
Input.txt



```
Hello Everyone

Hope Everything is      Ok



Iam Deepak R

Presently Studying in Ramaiah University (Deemed to be University) with Roll number "18ETCS002041"
```

Fig1.2 Input which is Read by the Program

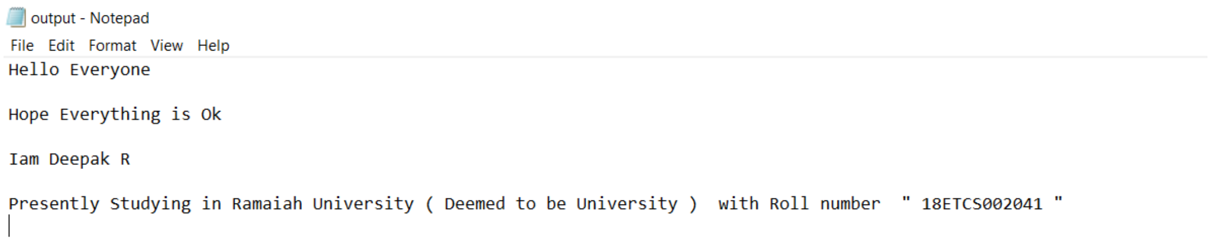
Output for the Given Problem



```
PS C:\Users\HP\Desktop\New folder (3)> lex lex1.l
PS C:\Users\HP\Desktop\New folder (3)> gcc lex.yy.c
PS C:\Users\HP\Desktop\New folder (3)> ./a.exe

lines = 5
Spaces = 25
```

Fig 1.3 Command Given to Produce Desired Output so to get number of lines and spaces

Output.txt



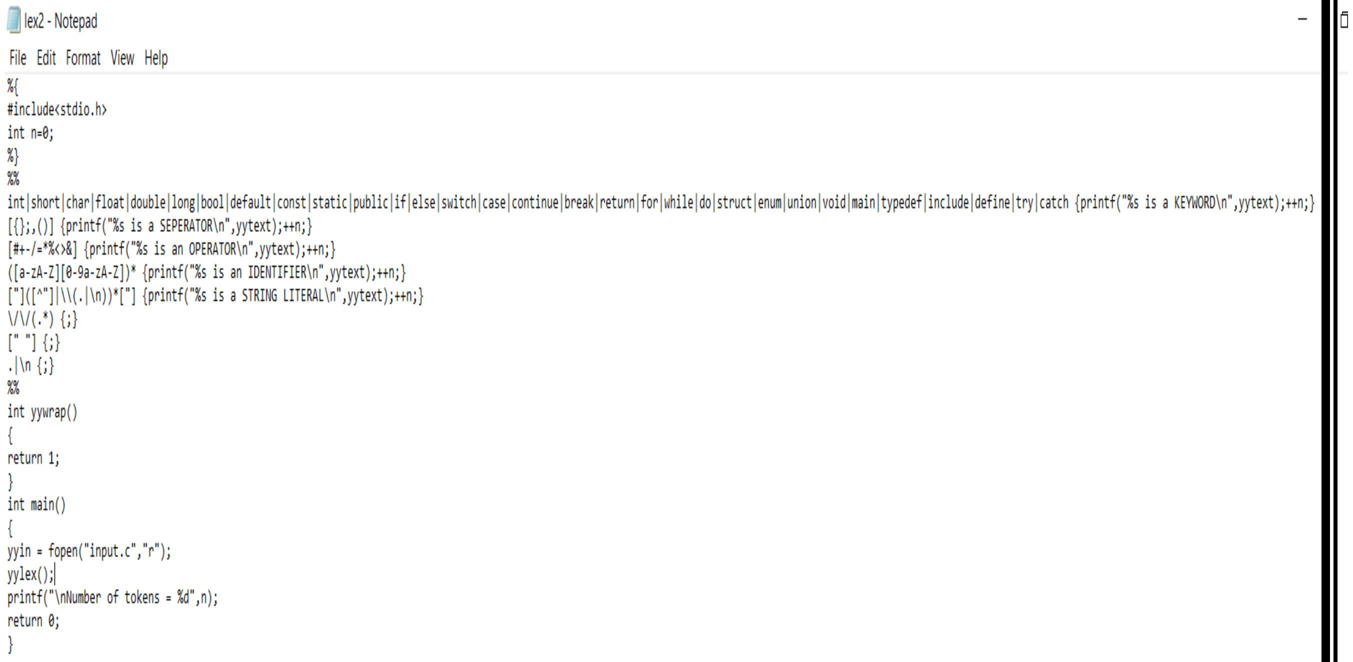
```
Hello Everyone

Hope Everything is Ok

Iam Deepak R

Presently Studying in Ramaiah University ( Deemed to be University )  with Roll number  " 18ETCS002041 "
```

Here

- **Multiple consecutive blank lines are compressed and number of lines at the end of given text is calculated so total number of lines are 5 which was 9 before Compression**
- **Multiple consecutive spaces are compressed and number of blank space at the end of given text so total spaces after compression is 25 which was 29 before compression.**
  - **Space before and after punctuation and after opening and closing parentheses are given in above output space is given after inverted commas and Parenthesis.**

Solution for Part A.2

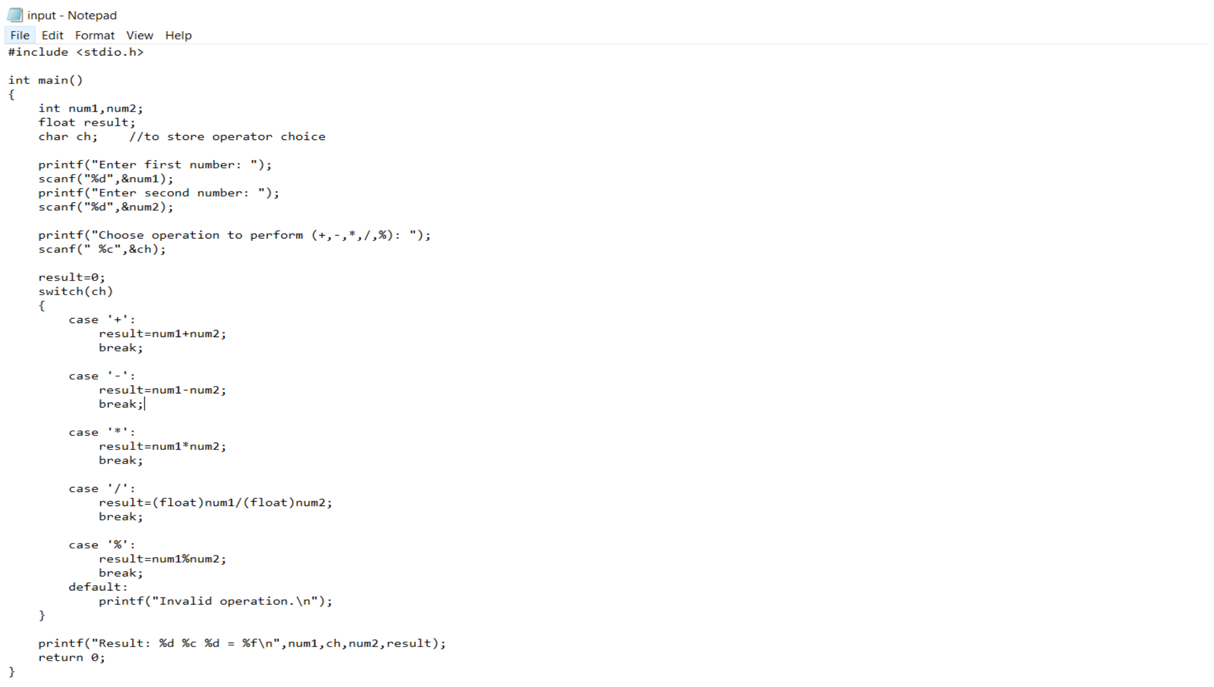
Lex2.l



```
%{
#include<stdio.h>
int n=0;
%}
%%
int|short|char|float|double|long|bool|default|const|static|public|if|else|switch|case|continue|break|return|for|while|do|struct|enum|union|void|main|typedef|include|define|try|catch {printf("%s is a KEYWORD\n",yytext);++n;}
[{};,()] {printf("%s is a SEPERATOR\n",yytext);++n;}
[#+-/=*%<>&] {printf("%s is an OPERATOR\n",yytext);++n;}
([a-zA-Z][0-9a-zA-Z])* {printf("%s is an IDENTIFIER\n",yytext);++n;}
["]([^"]|\\(.|\n))*["] {printf("%s is a STRING LITERAL\n",yytext);++n;}
\/\/(.*) {;}
[" "] {;}
.|\n {;}
%%
int yywrap()
{
return 1;
}
int main()
{
yyin = fopen("input.c","r");
yylex();
printf("\nNumber of tokens = %d",n);
return 0;
}
```

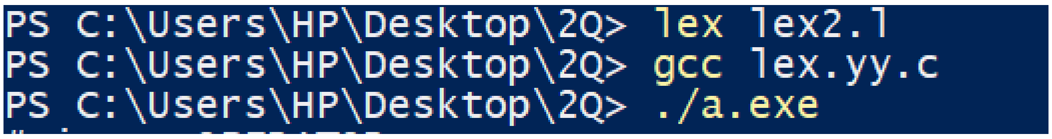
Fig2.1 Lex Program to Identify Tokens in the Given C Program.

Input.c



```
#include <stdio.h>

int main()
{
    int num1,num2;
    float result;
    char ch;    //to store operator choice

    printf("Enter first number: ");
    scanf("%d",&num1);
    printf("Enter second number: ");
    scanf("%d",&num2);

    printf("Choose operation to perform (+,-,*,/,%): ");
    scanf(" %c",&ch);

    result=0;
    switch(ch)
    {
        case '+':
            result=num1+num2;
            break;

        case '-':
            result=num1-num2;
            break;

        case '*':
            result=num1*num2;
            break;

        case '/':
            result=(float)num1/(float)num2;
            break;

        case '%':
            result=num1%num2;
            break;
        default:
            printf("Invalid operation.\n");
    }

    printf("Result: %d %c %d = %f\n",num1,ch,num2,result);
    return 0;
}
```

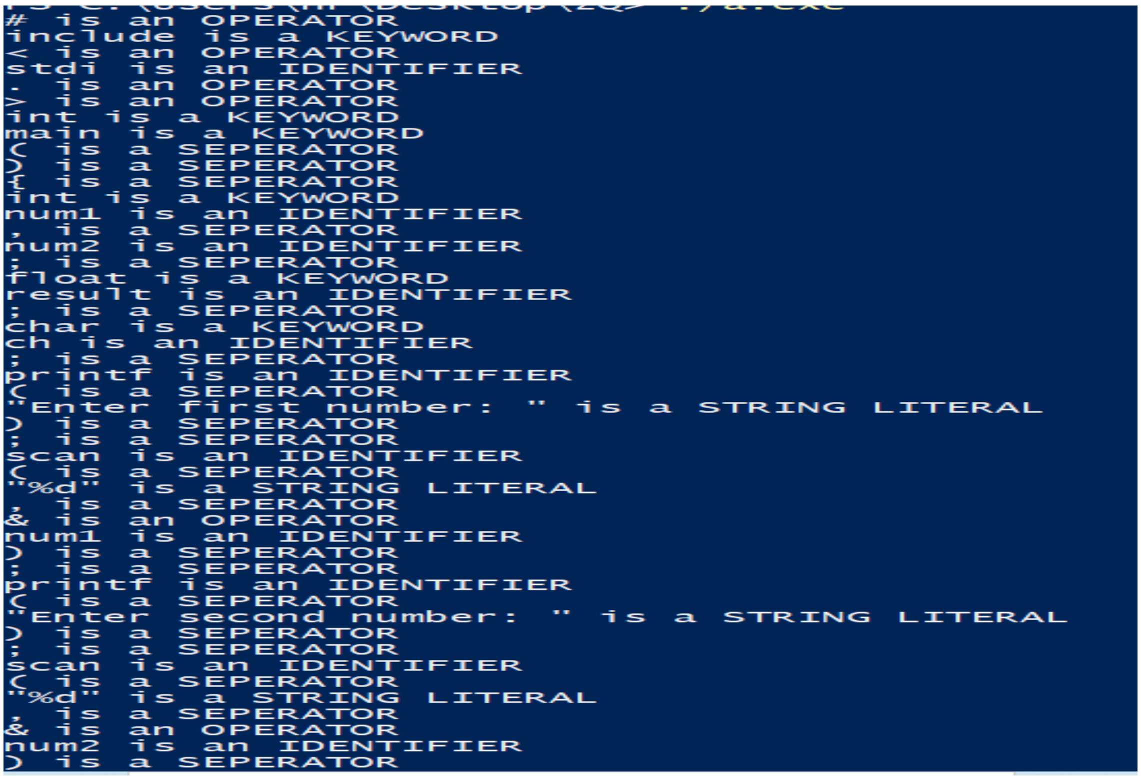
Fig2.2 Input which is Read by the Program(Given by Course leader)



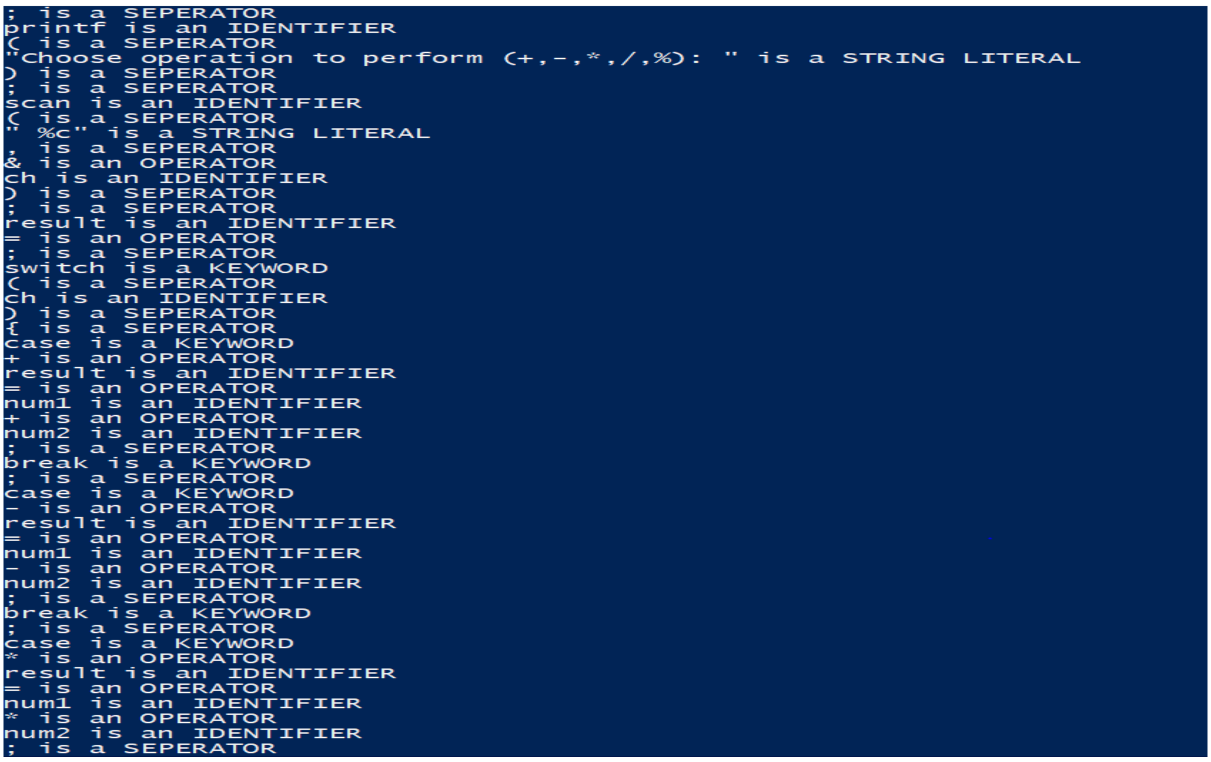
```
PS C:\Users\HP\Desktop\2Q> lex lex2.l
PS C:\Users\HP\Desktop\2Q> gcc lex.yy.c
PS C:\Users\HP\Desktop\2Q> ./a.exe
```

Fig 2.3 Command Given to Produce Desired Output

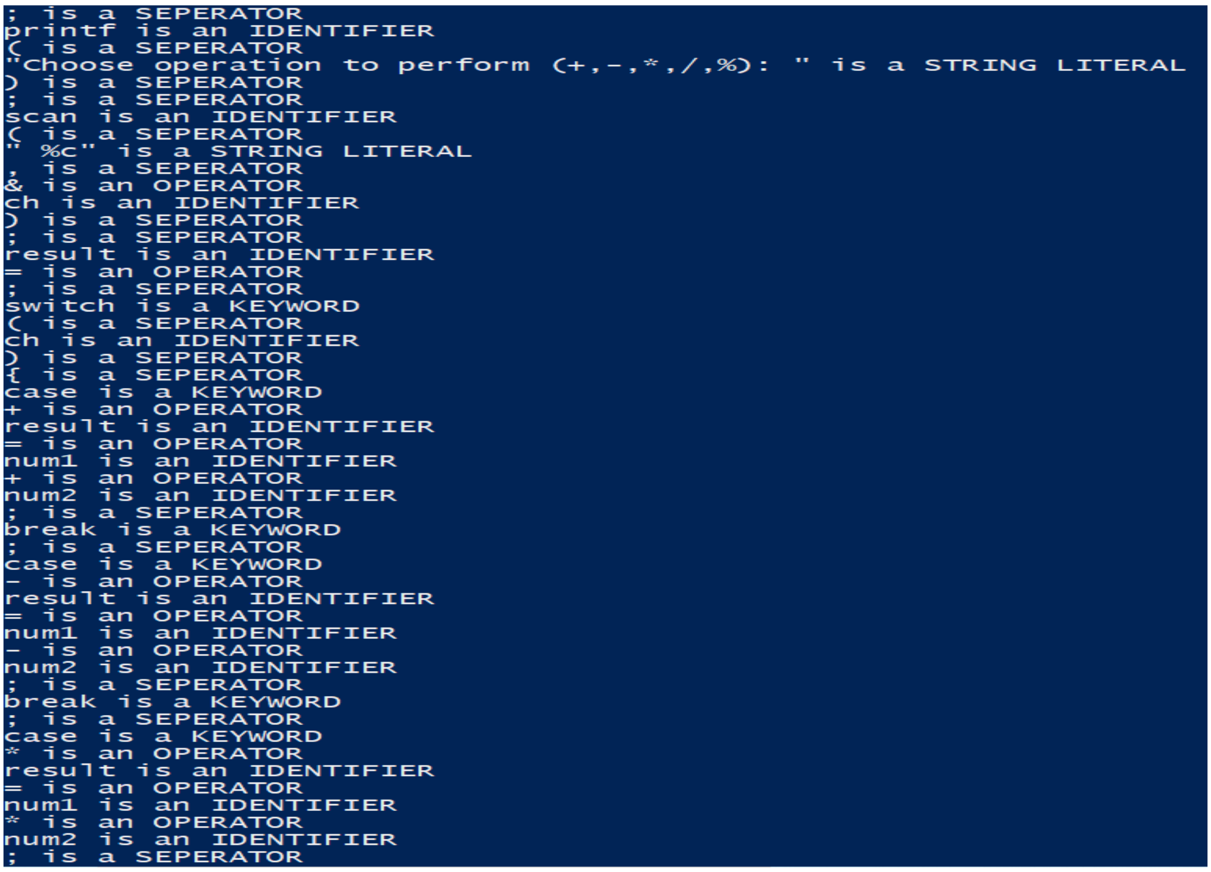
Output.



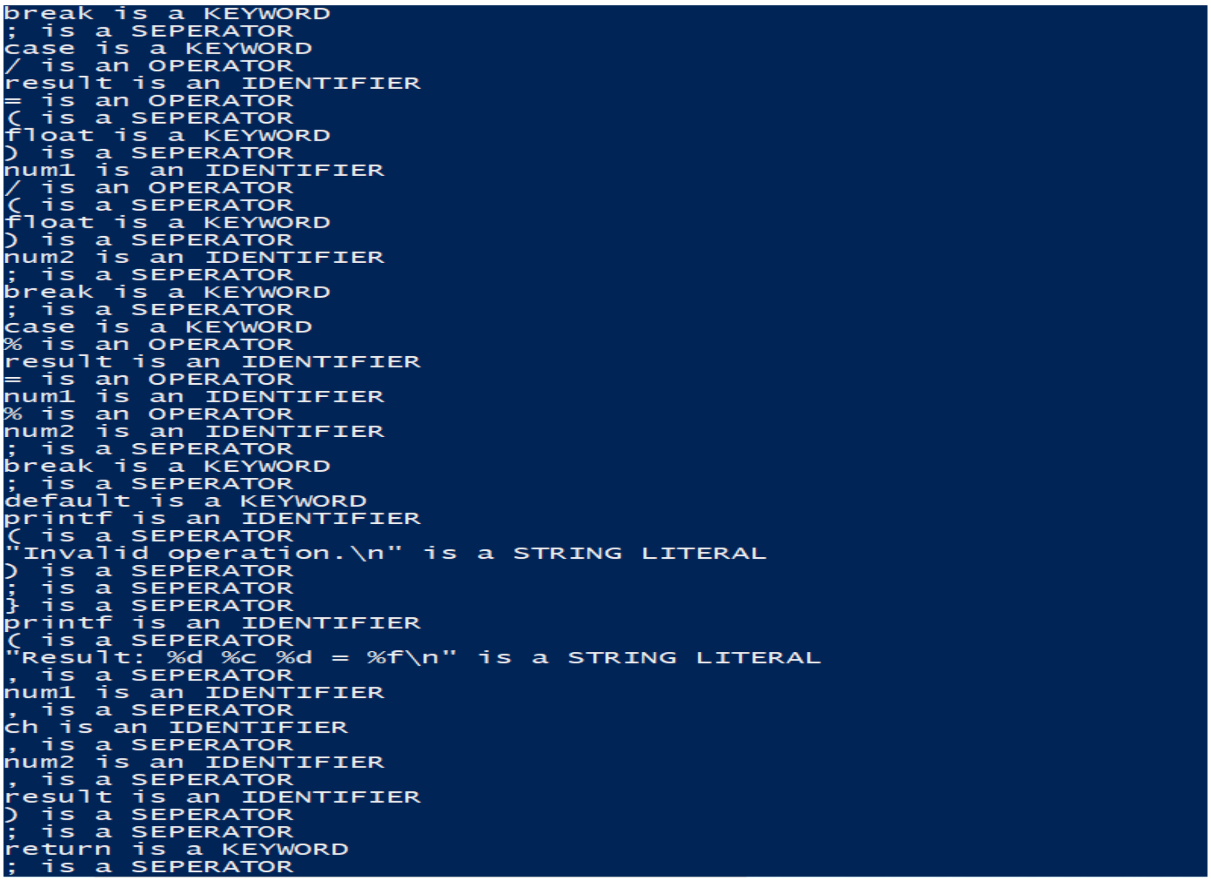
```
# is an OPERATOR
include is a KEYWORD
< is an OPERATOR
stdi is an IDENTIFIER
. is an OPERATOR
> is an OPERATOR
int is a KEYWORD
main is a KEYWORD
( is a SEPERATOR
) is a SEPERATOR
{ is a SEPERATOR
int is a KEYWORD
num1 is an IDENTIFIER
, is a SEPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
float is a KEYWORD
result is an IDENTIFIER
; is a SEPERATOR
char is a KEYWORD
ch is an IDENTIFIER
; is a SEPERATOR
printf is an IDENTIFIER
( is a SEPERATOR
"Enter first number: " is a STRING LITERAL
) is a SEPERATOR
; is a SEPERATOR
scan is an IDENTIFIER
( is a SEPERATOR
"%d" is a STRING LITERAL
, is a SEPERATOR
& is an OPERATOR
num1 is an IDENTIFIER
) is a SEPERATOR
; is a SEPERATOR
printf is an IDENTIFIER
( is a SEPERATOR
"Enter second number: " is a STRING LITERAL
) is a SEPERATOR
; is a SEPERATOR
scan is an IDENTIFIER
( is a SEPERATOR
"%d" is a STRING LITERAL
, is a SEPERATOR
& is an OPERATOR
num2 is an IDENTIFIER
) is a SEPERATOR
```



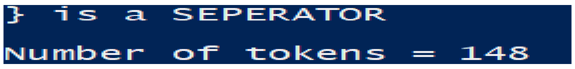
```
; is a SEPERATOR
printf is an IDENTIFIER
( is a SEPERATOR
"Choose operation to perform (+,-,*,/,%): " is a STRING LITERAL
) is a SEPERATOR
; is a SEPERATOR
scan is an IDENTIFIER
( is a SEPERATOR
" %c" is a STRING LITERAL
, is a SEPERATOR
& is an OPERATOR
ch is an IDENTIFIER
) is a SEPERATOR
; is a SEPERATOR
result is an IDENTIFIER
= is an OPERATOR
; is a SEPERATOR
switch is a KEYWORD
( is a SEPERATOR
ch is an IDENTIFIER
) is a SEPERATOR
{ is a SEPERATOR
case is a KEYWORD
+ is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
num1 is an IDENTIFIER
+ is an OPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
break is a KEYWORD
; is a SEPERATOR
case is a KEYWORD
- is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
num1 is an IDENTIFIER
- is an OPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
break is a KEYWORD
; is a SEPERATOR
case is a KEYWORD
* is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
num1 is an IDENTIFIER
* is an OPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
```



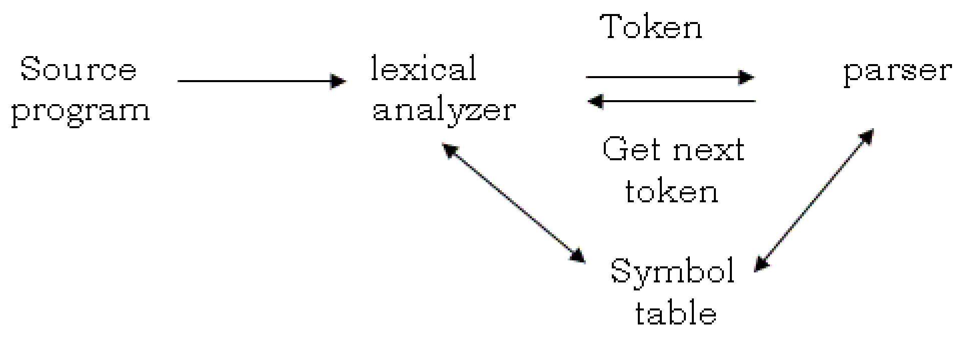
```
; is a SEPERATOR
printf is an IDENTIFIER
( is a SEPERATOR
"Choose operation to perform (+,-,*,/,%): " is a STRING LITERAL
) is a SEPERATOR
; is a SEPERATOR
scan is an IDENTIFIER
( is a SEPERATOR
" %c" is a STRING LITERAL
, is a SEPERATOR
& is an OPERATOR
ch is an IDENTIFIER
) is a SEPERATOR
; is a SEPERATOR
result is an IDENTIFIER
= is an OPERATOR
; is a SEPERATOR
switch is a KEYWORD
( is a SEPERATOR
ch is an IDENTIFIER
) is a SEPERATOR
{ is a SEPERATOR
case is a KEYWORD
+ is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
num1 is an IDENTIFIER
+ is an OPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
break is a KEYWORD
; is a SEPERATOR
case is a KEYWORD
- is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
num1 is an IDENTIFIER
- is an OPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
break is a KEYWORD
; is a SEPERATOR
case is a KEYWORD
* is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
num1 is an IDENTIFIER
* is an OPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
```



```
break is a KEYWORD
; is a SEPERATOR
case is a KEYWORD
/ is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
( is a SEPERATOR
float is a KEYWORD
) is a SEPERATOR
num1 is an IDENTIFIER
/ is an OPERATOR
( is a SEPERATOR
float is a KEYWORD
) is a SEPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
break is a KEYWORD
; is a SEPERATOR
case is a KEYWORD
% is an OPERATOR
result is an IDENTIFIER
= is an OPERATOR
num1 is an IDENTIFIER
% is an OPERATOR
num2 is an IDENTIFIER
; is a SEPERATOR
break is a KEYWORD
; is a SEPERATOR
default is a KEYWORD
printf is an IDENTIFIER
( is a SEPERATOR
"Invalid operation.\n" is a STRING LITERAL
) is a SEPERATOR
; is a SEPERATOR
} is a SEPERATOR
printf is an IDENTIFIER
( is a SEPERATOR
"Result: %d %c %d = %f\n" is a STRING LITERAL
, is a SEPERATOR
num1 is an IDENTIFIER
, is a SEPERATOR
ch is an IDENTIFIER
, is a SEPERATOR
num2 is an IDENTIFIER
, is a SEPERATOR
result is an IDENTIFIER
) is a SEPERATOR
; is a SEPERATOR
return is a KEYWORD
; is a SEPERATOR
```



```
} is a SEPERATOR
Number of tokens = 148
```



Steps Followed

1. Physical source file multibyte characters are mapped, in an implementation-defined manner, to the source character set (introducing new-line characters for end-of-line indicators) if necessary. Trigraph sequences are replaced by corresponding single-character internal representations.
2. Each instance of a backslash character (\) immediately followed by a new-line character is deleted, splicing physical source lines to form logical source lines. Only the last backslash on any physical source line shall be eligible for being part of such a splice. A source file that is not empty shall end in a new-line character, which shall not be immediately preceded by a backslash character before any such splicing takes place.
3. The source file is decomposed into preprocessing tokens and sequences of white-space characters (including comments). A source file shall not end in a partial preprocessing token or in a partial comment. Each comment is replaced by one space character. New-line characters are retained. Whether each nonempty sequence of white-space characters other than new-line is retained or replaced by one space character is implementation-defined.
4. Preprocessing directives are executed, macro invocations are expanded, and _Pragma unary operator expressions are executed. If a character sequence that matches the syntax of a universal character name is produced by token concatenation the behavior is undefined. A #include preprocessing directive causes the named header or source file to be processed from phase 1 through phase 4, recursively. All preprocessing directives are then deleted.
5. Each source character set member and escape sequence in character constants and string literals is converted to the corresponding member of the execution character set; if there is no corresponding member, it is converted to an implementation-defined member other than the null (wide) character.
6. Adjacent string literal tokens are concatenated.
7. White-space characters separating tokens are no longer significant. Each preprocessing token is converted into a token. The

resulting tokens are syntactically and semantically analyzed and translated as a translation unit.

8. All external object and function references are resolved. Library components are linked to satisfy external references to functions and objects not defined in the current translation. All such translator output is collected into a program image which contains information needed for execution in its execution environment.

References

Compilers: Principles, Techniques and Tools” by Alfred V Aho and Ravi Sethi