

|  |  |
| --- | --- |
| **ASSIGNMENT** | |
| **Course Code** | 19CSC305A |
| **Course Name** | Compilers |
| **Programme** | B. Tech. |
| **Department** | Computer Science and Engineering |
| **Faculty** | Faculty of Engineering & Technology |

#### 

|  |  |
| --- | --- |
| **Name of the Student** | Subhendu Maji |
| **Reg. No** | 18ETCS002121 |
| **Semester/Year** | 5TH semester / 2018 batch |
| **Course Leader/s** | Ms. Suvidha |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaration Sheet** | | | | | | | | |
| Student Name | Subhendu Maji | | | | | | | |
| Reg. No | 18ETCS002121 | | | | | | | |
| Programme | B. Tech. | | | | | Semester/Year | 5th sem / 2018 batch | |
| Course Code | 19CSC305A | | | | | | | |
| Course Title | Compilers | | | | | | | |
| Course Date |  | | to | |  | | | |
| Course Leader | Ms. Suvidha | | | | | | | |
| **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 |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | Signature of the Reviewer and date | | | | |
|  | | | |  | | | | |

# **Contents**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[**Declaration Sheet** ii](#_Toc58526280)

[**Contents** iii](#_Toc58526281)

[**Marking Scheme** 4](#_Toc58526282)

[**Question No. 1** 5](#_Toc58526283)

[1.1 Implementation in Lex 5](#_Toc58526284)

[1.2 Results and Comments 9](#_Toc58526285)

[**Question No. 2** 12](#_Toc58526286)

[2.1 Implementation in Lex 12](#_Toc58526287)

[2.2 Results and Comments 16](#_Toc58526288)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Assignment** | | | | |  |  | | |
| **Register No.** | | | **18ETCS002121** | **Name of the Student** | | **SUBHENDU MAJI** | | |
| **Sections** |  | **Marking Scheme** | | |  | **Marks** | | |
| **Max Marks** | **First**  **Examiner Moderator Marks** | | |
| **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** | | | | |

# **Question No. 1**

**Solution to Question No. 1:**

## Implementation in Lex

|  |  |
| --- | --- |
| **Token** | **Regular Expression** |
| NEW\_LINE | : \r\n |

1a. Multiple consecutive blank lines should be compressed and calculate number of lines at the end of given text.

%{

    #include <stdio.h>

    int lctr=0;

%}

*// defining regex*

new\_line [\r\n]

%%

{new\_line}    {lctr++;}

.\r\n    {fprintf(yyout, "%s",yytext);lctr++;}

%%

int yywrap(){}

int main()

{

    extern FILE \*yyin, \*yyout;

    yyin = fopen("input.c", "r");

    yyout = fopen("output1a.txt", "w");

    yylex();

    fclose(yyin);

    fclose(yyout);

    printf("number of newlines : %d\n",lctr);

    return 0;

}

Figure 1 Lex program of 1a.l

1.b Multiple consecutive spaces should be compressed and calculate number of blank space at the end of given text.

|  |  |
| --- | --- |
| **Token** | **Regular Expression** |
| SPACES | : [ ]+ |

%{

    #include <stdio.h>

    int space\_count=0;

%}

*// defining regex*

space [ ]+

%%

{space}    {space\_count++;fprintf(yyout," ");}

. {fprintf(yyout, "%s",yytext);}

%%

int yywrap(){}

int main()

{

    extern FILE \*yyin, \*yyout;

    yyin = fopen("input.c", "r");

    yyout = fopen("output1b.txt", "w");

    yylex();

    fclose(yyin);

    fclose(yyout);

    printf("compressed count of spaces : %d\n",space\_count);

    return 0;

}

Figure 2 Lex Program of 1b.l

1c. Space before and after punctuation and after opening and closing parentheses and also calculting no. of tokens.

%{

    #include <stdio.h>

    int tokens = 0;

%}

%%

*/\* remove white spaces\*/*

[ ]+    {fprintf(yyout," ");}

*/\*count identifiers and keywords\*/*

[a-zA-Z\_][a-zA-Z0-9\_]\*   {tokens++;fprintf(yyout,"%s",yytext);}

*/\*count operators\*/*

"<="|"=="|"="|"++"|"-"|"\*"|"+"|"&"|">"|"<"|"|"|"/"|"%"    {tokens++;fprintf(yyout,"%s",yytext);}

*/\*count and adding space before and after puncutations\*/*

[,;:]    {tokens++;fprintf(yyout," %s ",yytext);}

[,;:][\n]    {tokens++;fprintf(yyout," %s",yytext);}

*/\*count and adding space after opening brackets \*/*

[({\[]    {tokens++;fprintf(yyout,"%s ",yytext);}

[({\[][\n]    {tokens++;fprintf(yyout,"%s ",yytext);}

*/\*count and adding space before closing brackets\*/*

[)}\]]    {tokens++;fprintf(yyout," %s",yytext);}

[)}\]][\n]    {tokens++;fprintf(yyout," %s",yytext);}

*/\*count float literal\*/*

[0-9]\*"."[0-9]+    {tokens++;fprintf(yyout,"%s",yytext);}

*/\*count int literal\*/*

[0-9]+   {tokens++;fprintf(yyout,"%s",yytext);}

*/\*count string literal\*/*

\"(\\.|[^"\\])\*\"    {tokens++;fprintf(yyout,"%s",yytext);}

*/\*count character literal\*/*

\'(\\.|[^'\\])\*\'    {tokens++;fprintf(yyout,"%s",yytext);}

*/\*skip pre-processor \*/*

#[a-zA-Z]\* ;

"<"+([a-zA-Z]\*"."[a-zA-Z]\*)+">" ;

*/\*skip comments\*/*

\/\/(.\*)    ;

*/\*skip newlines\*/*

[\n]    ;

.\n    {fprintf(yyout, "%s",yytext);}

%%

int yywrap(){}

int main()

{

    extern FILE \*yyin, \*yyout;

    yyin = fopen("input.c", "r");

    yyout = fopen("output1c.txt", "w");

    yylex();

    fclose(yyin);

    fclose(yyout);

    printf("number of tokens : %d\n",tokens);

    return 0;

}

Figure 3 lex program of 1c.l

## 1.2 Results and Comments

Note: I am using the same *input.c* file for all the three lex program.



Figure 4 Input file

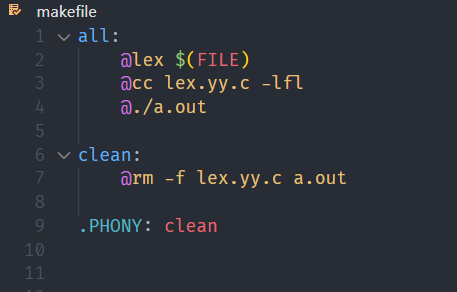


Figure 5 makefile

1a. Executing lex program to compress multiple consecutive blank lines and calculate number of lines at the end of given text.

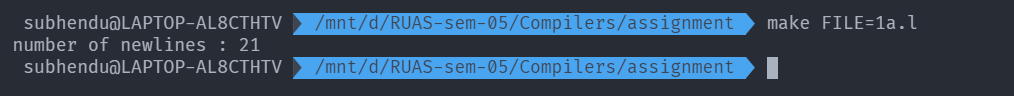


Figure 6 executing 1a.l

Output file after execution:

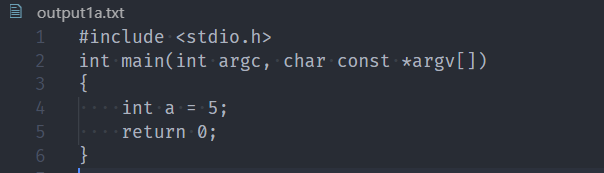


Figure 7 output file after execution

As we can see in above figure (Fig 7), multiple lines are compressed and stored in output1a.txt file.

1b. Executing lex program to compress multiple consecutive spaces and calculate number of spaces at the end of given text.

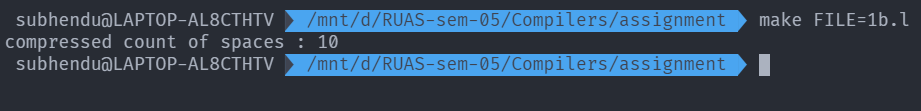


Figure 8 execution of 1b.l

Output file after execution:

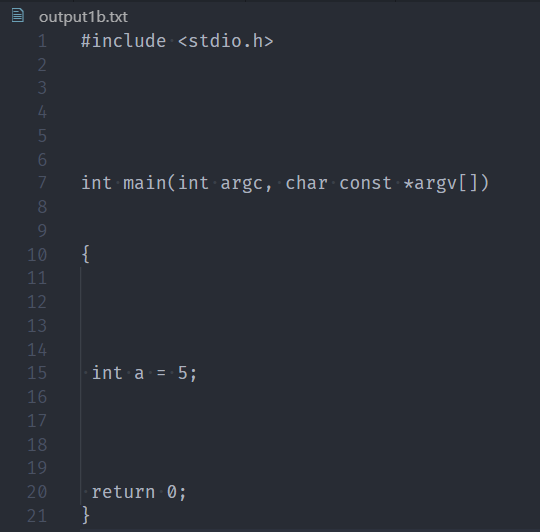


Figure 9 output file after execution

As we can see in above figure (Fig 9), multiple spaces are compressed and stored in output1b.txt file.

1c. Space before and after punctuation and after opening and closing parentheses and also calculting no. of tokens.

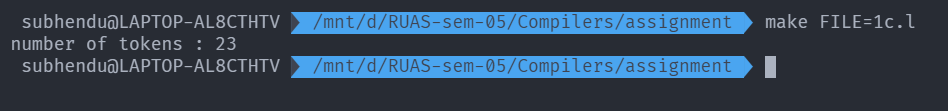


Figure 10 execution of 1c.l

Output file after execution:

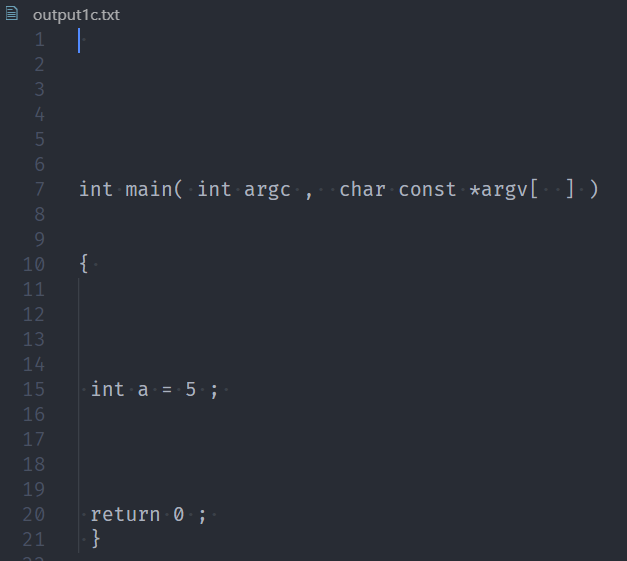


Figure 11 output file after execution

As we can see in above figure (Fig 11), Space before and after punctuation and after opening and closing parentheses is added and stored in output1c.txt file

# **Question No. 2**

**Solution to Question No. 2:**

## 2.1 Implementation in Lex

The indivisible unit of a program is called a token. A compiler breaks the program into the smallest possible units and proceeds to the various stages of the compilation. Such units are tokens. As seen here, tokens are the units in which compilers act on. Typically, in a language like C, there are the following tokens: Identifiers, Keywords, Constants, Strings, Operators and Special Symbols.

**Regular Expression**

The regular expressions that are used to match and produce the various token are straight forward. For the keyword token, the regular expressions are the keywords themselves.

|  |  |
| --- | --- |
| **Token** | **Regular Expression** |
| KEYWORD | auto|double|int|struct|break|else|long|switch|case|typedef|char|extern|return|continue|for|signed|void|do|if|static|while|default|goto|sizeof|const|float|short |

For the identifier token, any set of alphanumeric characters that start with an alphabet is assumed to be an identifier. Thus, the regular expression for this would be:

|  |  |
| --- | --- |
| **Token** | **Regular Expression** |
| IDENTIFIER | : [a-zA-Z]+[a-zA-Z0-9]\* |

For the integer number token, the regular expression would be to match an optional negative sign, followed by any set of digits. For the floating-point number token on the other hand, there are two cases to be handled. One is the case where the number is defined in terms of regular digits and a decimal point, the other, when the number is defined using scientific notation. Thus, the regular expressions are:

|  |  |
| --- | --- |
| **Token** | **Regular Expression** |
| INT\_NUM | : -?[0-9]+ |
| FLOAT\_NUM | : -?([0-9]+\.[0-9]+)([eE][-+]?[0-9]+)? |
| STRING\_LITERAL | : \"[^\"]\*\" |
| CHAR\_LITERAL | : \'[^\']\*\' |

The tokens for the special symbols and operators are matched using the characters themselves, with the characters escaped when necessary.

|  |  |
| --- | --- |
| **Token** | **Regular Expression** |
| BLOCK\_START | : \{ |
| BLOCK\_END | : \} |
| OPEN\_BRAC | : \( |
| CLOSE\_BRAC | : \) |
| SEMICOLON | : \; |
| COMMA | : \, |
| COLON | : \: |
|  |  |
| PLUS | : \+ |
| MINUS | : \- |
| MUL | : \\* |
| DIV | : \/ |
| MOD | : % |
| INC | : \+\+ |
| DEC | : \-\- |
| ASSIGNMENT | : = |
| EQ | : == |
| LT | : < |
| LEQ | : <= |
| GT | : > |
| GEQ | : >= |
| AND | : && |
| OR | : \|\| |
| NOT | : ! |

The regex to ignore any single line comment.

|  |  |
| --- | --- |
| **Token** | **Regular Expression** |
| COMMENT | : \/\/(.\*) |

The following is the implementation of the previously mentioned tokens and regular expressions as a lex program.

Program:

%{

    int count=0;

%}

*// defining regex*

*// literals*

int\_num -?[0-9]+

float\_num -?([0-9]+\.[0-9]+)([eE][-+]?[0-9]+)?

string\_literal \"[^\"]\*\"

char\_literal \'[^\']\*\'

*// variable*

identifier [a-zA-Z\_][a-zA-Z0-9\_]\*

*// special symbols*

block\_start \{

block\_end \}

open\_brac \(

close\_brac \)

semicolon \;

comma \,

colon \:

*// operators*

plus \+

minus \-

mul \\*

div \/

mod %

inc \+\+

dec \-\-

neq !=

assignment =

eq ==

lt <

leq <=

gt >

geq >=

and &&

or \|\|

not !

bitand &

*//single line comment*

comment \/\/(.\*)

*// preprocessors*

preprocessor "#include <stdio.h>"

%%

auto|double|int|struct|break|else|long|switch|case|typedef|char|extern|return|continue|for|signed|void|do|if|static|while|default|goto|sizeof|const|float|short    %{count++; fprintf(yyout, "\n%d - keywords: %s", count,yytext);%}

{identifier}    %{count++;fprintf(yyout, "\n%d - identifier: %s",count, yytext);%}

{int\_num}|{float\_num}|{string\_literal}|{char\_literal}   %{count++; fprintf(yyout, "\n%d - literal: %s",count, yytext);%}

{block\_start}|{block\_end}|{open\_brac}|{close\_brac}|{semicolon}|{comma}|{colon}    %{count++; fprintf(yyout, "\n%d - seperator: %s",count, yytext);%}

{plus}|{minus}|{mul}|{div}|{mod}|{inc}|{dec}|{neq}|{assignment}|{eq}|{lt}|{leq}|{gt}|{geq}|{and}|{or}|{not}|{bitand}    %{count++; fprintf(yyout, "\n%d - operator: %s", count,yytext);%}

{comment}    ;

{preprocessor}     ;

.    ;

%%

int yywrap() {

    return 1;

}

int main()

{

    extern FILE\* yyin;

    yyin = fopen("input.c", "r");

    yyout = fopen("Output.txt", "w");

    yylex();

    fclose(yyin);

    fclose(yyout);

    printf("\n Total no of token = %d\n", count);

}

Figure 12 Lex program source Code

## 2.2 Results and Comments

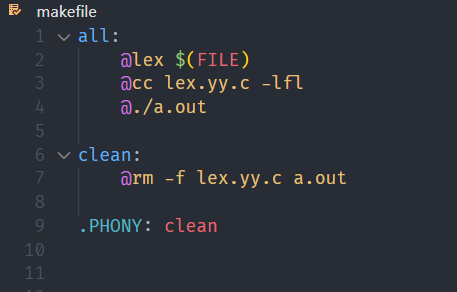


Figure 13 makefile

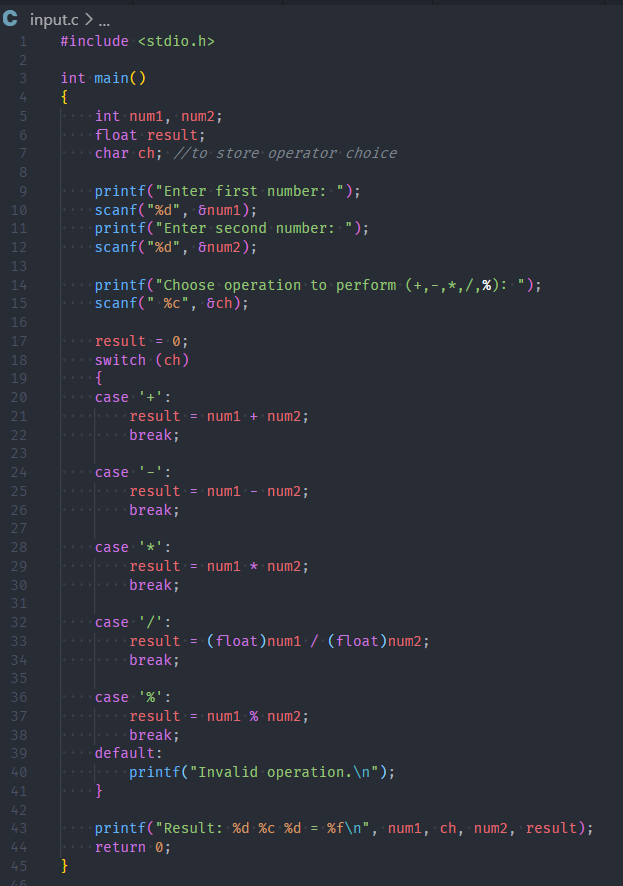


Figure 4 given input file in question

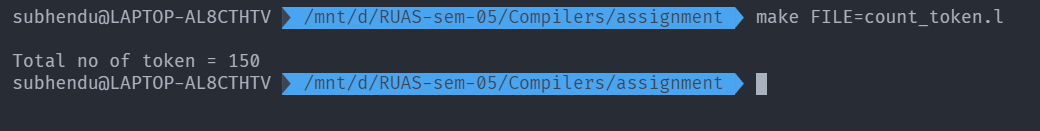
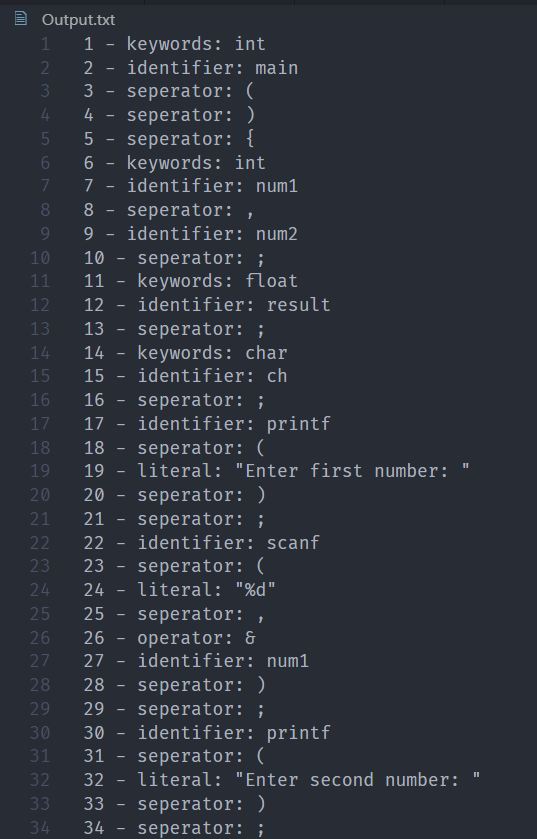
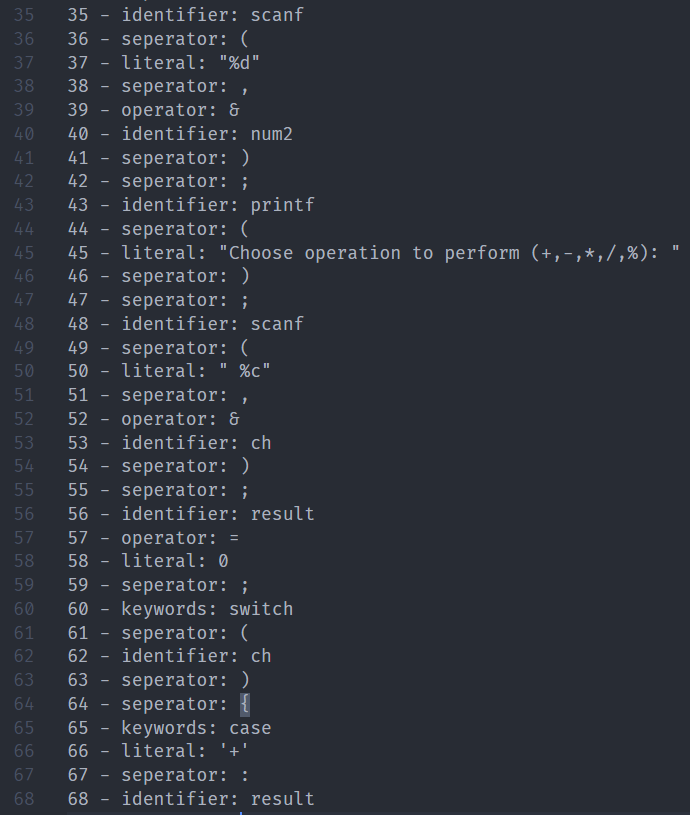
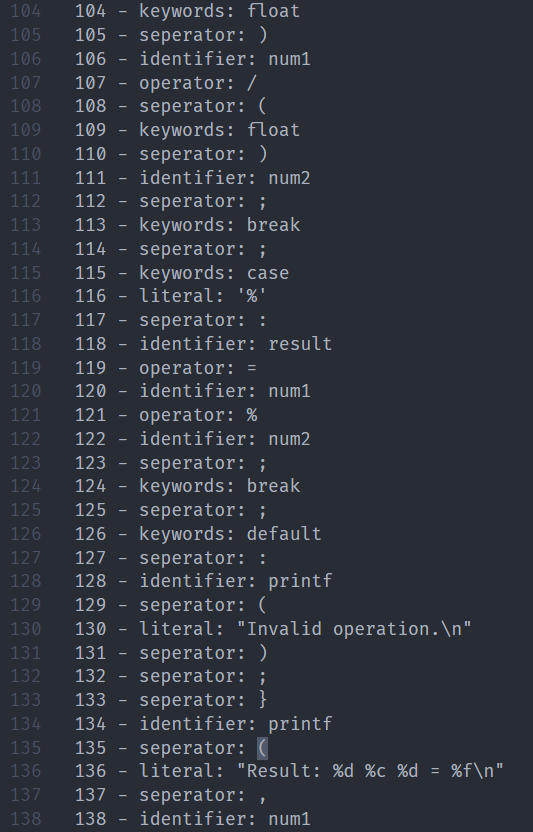
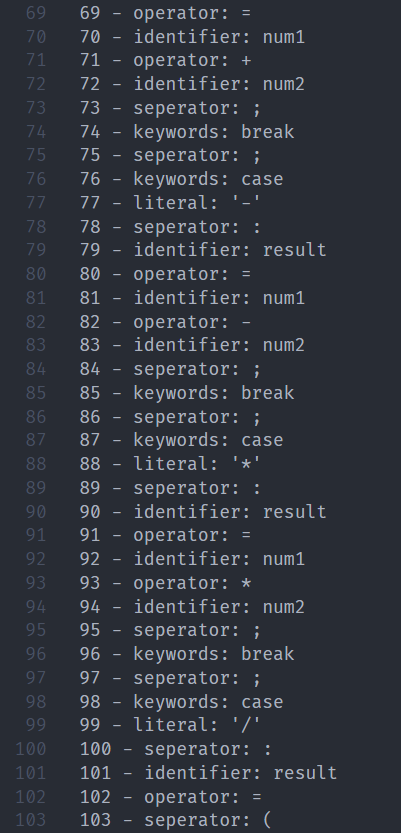


Figure 15 execution of count\_token.l

Output file in which generated token isstored:



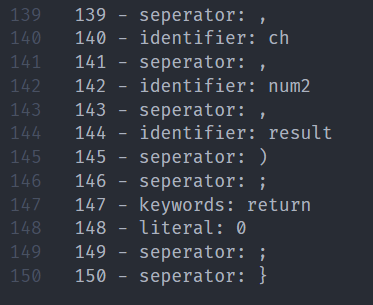


Figure 16 Output file in which token generated

Therefore, to verify the above output I calculated the token manually:

|  |  |
| --- | --- |
| Serial Number | Token |
| 1 | int |
| 2 | main |
| 3 | ( |
| 4 | ) |
| 5 | { |
| 6 | int |
| 7 | num1 |
| 8 | , |
| 9 | num2 |
| 10 | ; |
| 11 | float |
| 12 | result |
| 13 | ; |
| 14 | char |
| 15 | ch |
| 16 | ; |
| 17 | printf |
| 18 | ( |
| 19 | "Enter first number: " |
| 20 | ) |
| 21 | ; |
| 22 | scanf |
| 23 | ( |
| 24 | "%d" |
| 25 | , |
| 26 | & |
| 27 | num1 |
| 28 | ) |
| 29 | ; |
| 30 | printf |
| 31 | ( |
| 32 | "Enter second number: " |
| 33 | ) |
| 34 | ; |
| 35 | scanf |
| 36 | ( |
| 37 | "%d" |
| 38 | , |
| 39 | & |
| 40 | num2 |
| 41 | ) |
| 42 | ; |
| 43 | printf |
| 44 | ( |
| 45 | "Choose operation to perform (+,-,\*,/,%): " |
| 46 | ) |
| 47 | ; |
| 48 | scanf |
| 49 | ( |
| 50 | " %c" |
| 51 | , |
| 52 | & |
| 53 | ch |
| 54 | ) |
| 55 | ; |
| 56 | result |
| 57 | = |
| 58 | 0 |
| 59 | ; |
| 60 | switch |
| 61 | ( |
| 62 | ch |
| 63 | ) |
| 64 | { |
| 65 | case |
| 66 | '+' |
| 67 | : |
| 68 | result |
| 69 | = |
| 70 | num1 |
| 71 | + |
| 72 | num2 |
| 73 | ; |
| 74 | break |
| 75 | ; |
| 76 | case |
| 77 | '-' |
| 78 | : |
| 79 | result |
| 80 | = |
| 81 | num1 |
| 82 | - |
| 83 | num2 |
| 84 | ; |
| 85 | break |
| 86 | ; |
| 87 | case |
| 88 | '\*' |
| 89 | : |
| 90 | result |
| 91 | = |
| 92 | num1 |
| 93 | \* |
| 94 | num2 |
| 95 | ; |
| 96 | break |
| 97 | ; |
| 98 | case |
| 99 | '/' |
| 100 | : |
| 101 | result |
| 102 | = |
| 103 | ( |
| 104 | float |
| 105 | ) |
| 106 | num1 |
| 107 | / |
| 108 | ( |
| 109 | float |
| 110 | ) |
| 111 | num2 |
| 112 | ; |
| 113 | break |
| 114 | ; |
| 115 | case |
| 116 | '%' |
| 117 | : |
| 118 | result |
| 119 | = |
| 120 | num1 |
| 121 | % |
| 122 | num2 |
| 123 | ; |
| 124 | break |
| 125 | ; |
| 126 | default |
| 127 | : |
| 128 | printf |
| 129 | ( |
| 130 | "Invalid operation.\n" |
| 131 | ) |
| 132 | ; |
| 133 | } |
| 134 | printf |
| 135 | ( |
| 136 | "Result: %d %c %d = %f\n" |
| 137 | , |
| 138 | num1 |
| 139 | , |
| 140 | ch |
| 141 | , |
| 142 | num2 |
| 143 | , |
| 144 | result |
| 145 | ) |
| 146 | ; |
| 147 | return |
| 148 | 0 |
| 149 | ; |
| 150 | } |

Hence there are 150 tokens in the given C file. It concludes the lex program is working correctly.

**Bibliography**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. <https://www.geeksforgeeks.org/lex-program-to-take-input-from-file-and-remove-multiple-spaces-lines-and-tabs/>
2. <https://www.geeksforgeeks.org/lex-program-to-count-the-number-of-lines-spaces-and-tabs/?ref=rp>