**Name :** Sarthak Sengar **Roll no. :** 4130

**1. Write a Lex program to count the number of lines and characters in the input file.**

%{

  int lines = 0;

  int characters = 0;

%}

%%

[\n] { lines++; }

. { characters++; }

%%

int main() {

  yyin = fopen("input.txt", "r");

  yylex();

  printf("number of lines: %d\n", lines + 1);

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

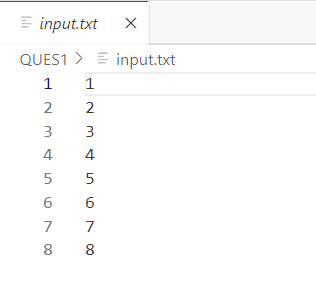
  return 1;

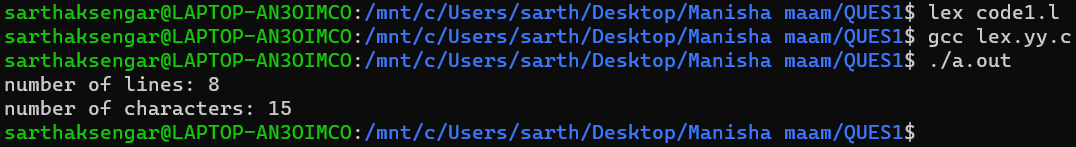
}

int yywrap() {

  return 1;

}



****

**2. Write a Lex program that implements the Caesar cipher: it replaces every letter with the one three letters after in alphabetical order, wrapping around at Z. e.g. a is replaced by d, b by e,and so on z by c.**

%{

  int rot = 0;

%}

%%

[A-Z] { fprintf(yyout, "%c", (yytext[0] - 'A' + rot) % 26 + 'A'); }

[a-z] { fprintf(yyout, "%c", (yytext[0] - 'a' + rot) % 26 + 'a'); }

. { fprintf(yyout, "%s", yytext); }

%%

int main(void) {

  printf("Enter Key (ROT): ");

  scanf("%d", &rot);

  yyin = fopen("input.txt", "r");

  yyout = fopen("output.txt", "w");

  yylex();

  fclose(yyin);

  fclose(yyout);

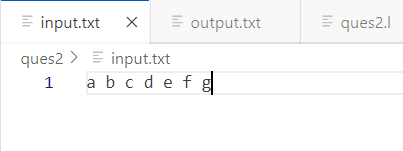
  return 0;

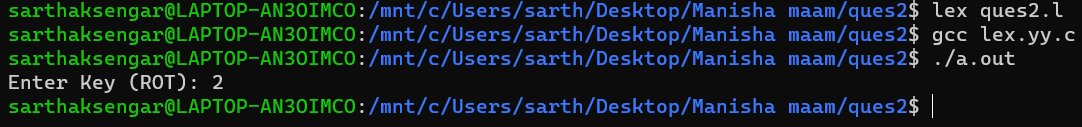
}

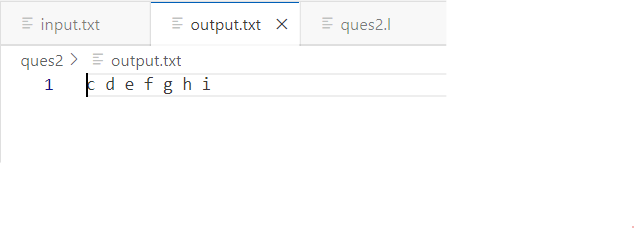
int yywrap() {

  return 1;

}



****

****

**3. Write a Lex program that finds the longest word (defined as a contiguous string of upper-and lower-case letters) in the input.**

%{

  int length = 0;

  char \*word = NULL;

%}

%%

[a-zA-Z]+ {

  if (yyleng > length) {

    length = yyleng;

    word = yytext;

  }

}

[ |\n|\r|\t] { ; }

. { ; }

%%

int main(void) {

  yyin = fopen("input.txt", "r");

  yylex();

  fclose(yyin);

  printf("Longest Word: %.\*s\n", length, word);

  printf("Length of Longest Word: %d\n", length);

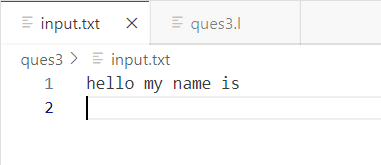
  return 0;

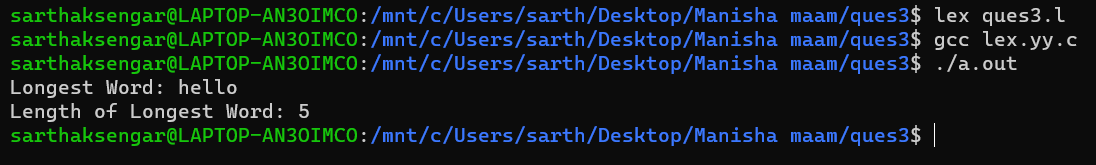
}

int yywrap() {

  return 1;

}

****

****

**4. Write a Lex program that distinguishes keywords, integers, floats, identifiers, operators, and comments in any simple programming language.**

%{

  int integers = 0;

  int floats = 0;

  int identifiers = 0;

  int operators = 0;

  int comments = 0;

%}

%%

[#].\* { ; } // preprocessor directives

[ |\n|\t] { ; } // whitespaces

[,|;|"("|")"|"{"|"}"|"\["|"\]"] { ; } // brackets, delimiters

"//".\* { comments++; printf("%s <- comment\n", yytext); } // single line comments

[0-9]+ { integers++; printf("%s <- integer\n", yytext); } // integers

[0-9]+("."[0-9]+) { floats++; printf("%s <- float\n", yytext); } // floats

void|int|float|main|char|for|while|continue|switch|case|break|if|else|return|true|false { printf("%s <- keyword\n", yytext); } // keywords

"<="|">="|"!="|"=="|"<"|">"|"&"|"|"|"^"|"<<"|">>"|"~"|"&&"|"||"|"!"|"++"|"--"|"="|"+"|"-"|"\*"|"/"|"%" { operators++; printf("%s <- operator\n", yytext); } // operators

[']([^\\\']|\\.)?['] { ; } // characters

["]([^\\\"]|\\.)\*["] { ; } // strings

[a-zA-Z\_]+[a-zA-Z0-9\_]\* { identifiers++; printf("%s <- identifier\n", yytext); } // identifiers

%%

int main() {

  yyin = fopen("input.c", "r");

  yylex();

  printf("\n");

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

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

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

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

  printf("number of comments: %d", comments);

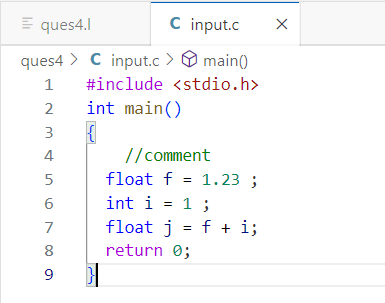
  return 0;

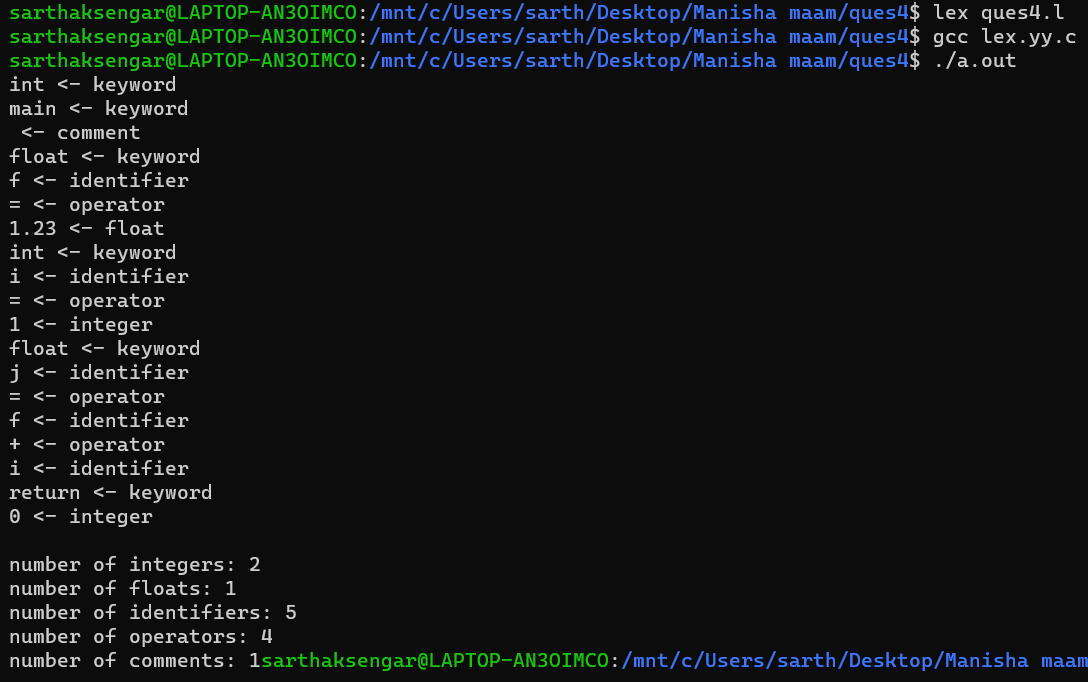
}

int yywrap() {

  return 1;

}





**5. Write a Lex program to count the number of identifiers in a C file.**

%{

    int identifiers = 0;

%}

%%

[#].\* { ; } // preprocessor directives

[ |\n|\t] { ; } // whitespaces

[,|;|"("|")"|"{"|"}"|"\["|"\]"] { ; } // brackets, delimiters

"//".\* { ; } // single line comment

-?[0-9]+("."[0-9]+)? { ; } // numbers

void|int|float|main|char|for|while|continue|switch|case|break|if|else|return|true|false { ; } // keywords

"<="|">="|"!="|"=="|"<"|">" { ; }  // relational operators

"&"|"|"|"^"|"<<"|">>"|"~" { ; }  // bitwise operators

"&&"|"||"|"!" { ; } // logical operators

"++"|"--" { ; } // postfix/prefix operators

"="|"+"|"-"|"\*"|"/"|"%" { ; } // other operators

[']([^\\\']|\\.)?['] { ; } // characters

["]([^\\\"]|\\.)\*["] { ; } // strings

[a-zA-Z\_]+[a-zA-Z0-9\_]\* { identifiers++; printf("%s <- identifier\n", yytext); } // identifiers

%%

int main() {

  yyin = fopen("prog.c", "r");

  yylex();

  printf("\nnumber of C identifiers: %d\n", identifiers);

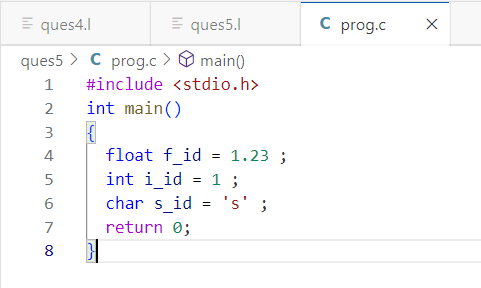
  return 0;

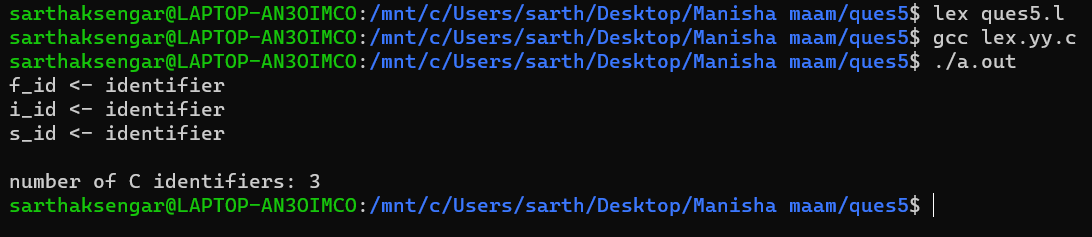
}

int yywrap() {

  return 1;

}

****

****

**6. Write a Lex program to count the number of words, characters, blank spaces and lines in a C file.**

%{

  int words = 0;

  int lines = 0;

  int spaces = 0;

  int characters = 0;

%}

%%

[^ \t\n,\.:;]+ { words++; characters += yyleng; }

[\n] { lines++; characters += yyleng; }

[ |\t] { spaces++; characters += yyleng; }

. { characters++; }

%%

int main() {

  yyin = fopen("prog.c", "r");

  yylex();

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

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

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

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

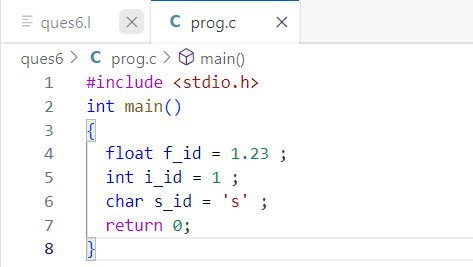
  return 0;

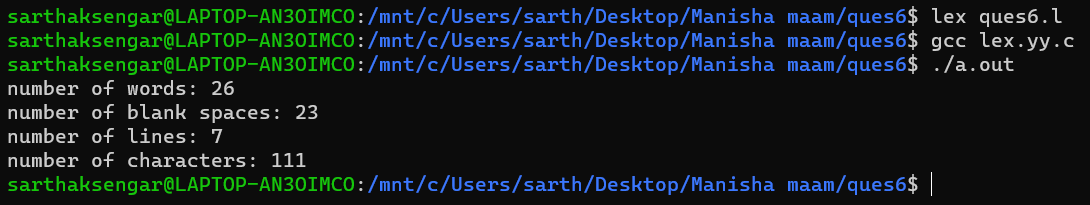
}

int yywrap() {

  return 1;

}

****

****

**7. Write a Lex specification program that generates a C program which takes a string “abcd” and prints the following output.**

**abcd**

**abc**

**ab**

**a**

%{

  #include <stdio.h>

%}

%%

[a-zA-Z]+ {

    for(int i =yyleng-1 ; i>=0 ; i--){

        for(int j = 0 ; j<= i ; j++ ){

            printf("%c", yytext[j]);

        }

        printf("\n");

    }

 }

.|\n { ; }

%%

int main() {

  yyin = fopen("input.txt", "r");

  yylex();

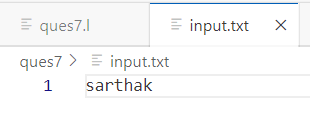
  return 0;

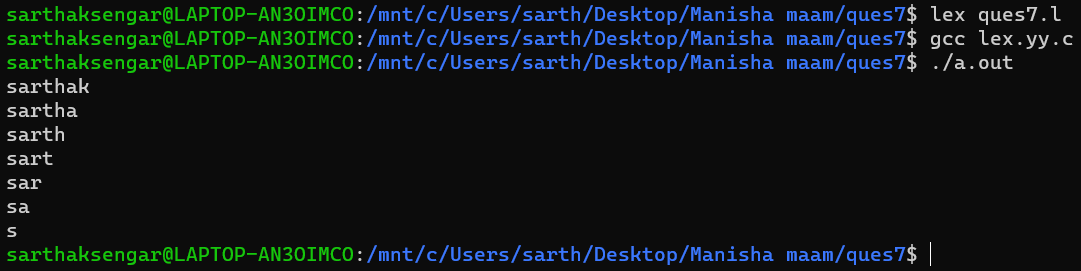
}

int yywrap() {

  return 1;

}

****

****

**8. A program in Lex to recognize a valid arithmetic expression.**

%{

#include<stdio.h>

#include<string.h>

int flag=0,i=0,j,k=0;

char operand[20][20],oparator[20][20];

%}

%%

[a-zA-Z0-9]+  {flag++; strcpy(operand[i],yytext);  i++;}

[-+\*/]  {flag--;  strcpy(oparator[k],yytext);    k++;}

%%

int main(int argc, char\* argv[])

{

    printf("enter an arithmetic expression\n");

    yylex();

    if(flag!=1)

        printf("Invalid expression\n");

    else

    {

        printf("Valid expression\n");

        printf("The operands are\t");

        for(j=0;j<i;j++)

            printf("%s\t",operand[j]);

        printf("\nThe operators are\t");

        for(j=0;j<k;j++)

            printf("%s\t",oparator[j]);

        printf("\n");

    }

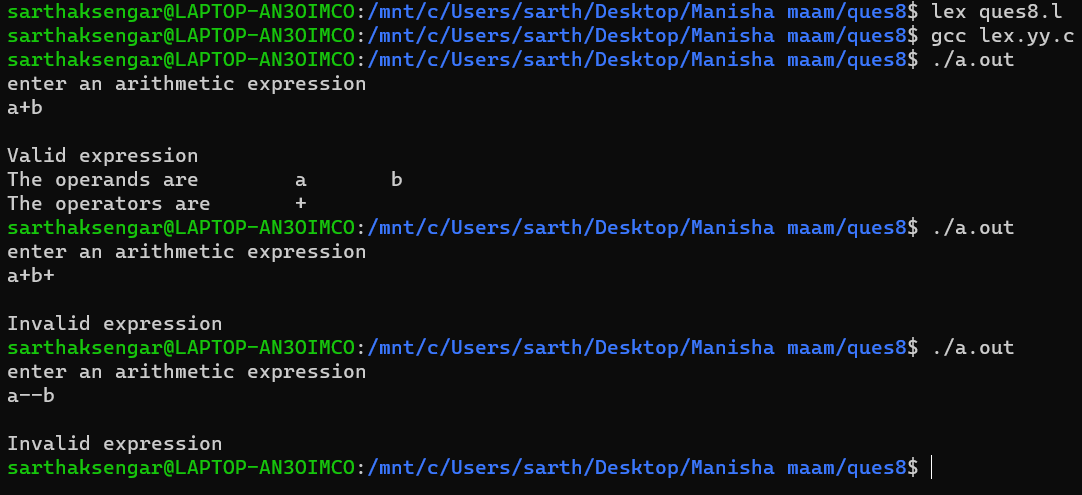
}

int yywrap( )

{

    return 1;

}

****

**9. Write a YACC program to find the validity of a given expression (for operators + - \* and /)**

**10. A Program in YACC which recognizes a valid variable which starts with letter followed by a digit. The letter should be in lowercase only.**

**11. A Program in YACC to evaluate an expression (simple calculator program for addition and subtraction, multiplication, division).**

**12. Program in YACC to recognize the strings “ab”, “aabb”, ”aaabbb”,... of the language (an bn,n>=1).**

**13. Program in YACC to recognize the language (anb , n>=10). (Output to say input is valid or not)**