**DAY 1**

**EXPERIMENT-1**: LEX program to identify the capital words from the given input.

%{

%}

%%

[A-Z]+[\t\n ] { printf("%s is a capital word\n",yytext); }

.|\n ;

%%

int yywrap( )

{

return 1;

}

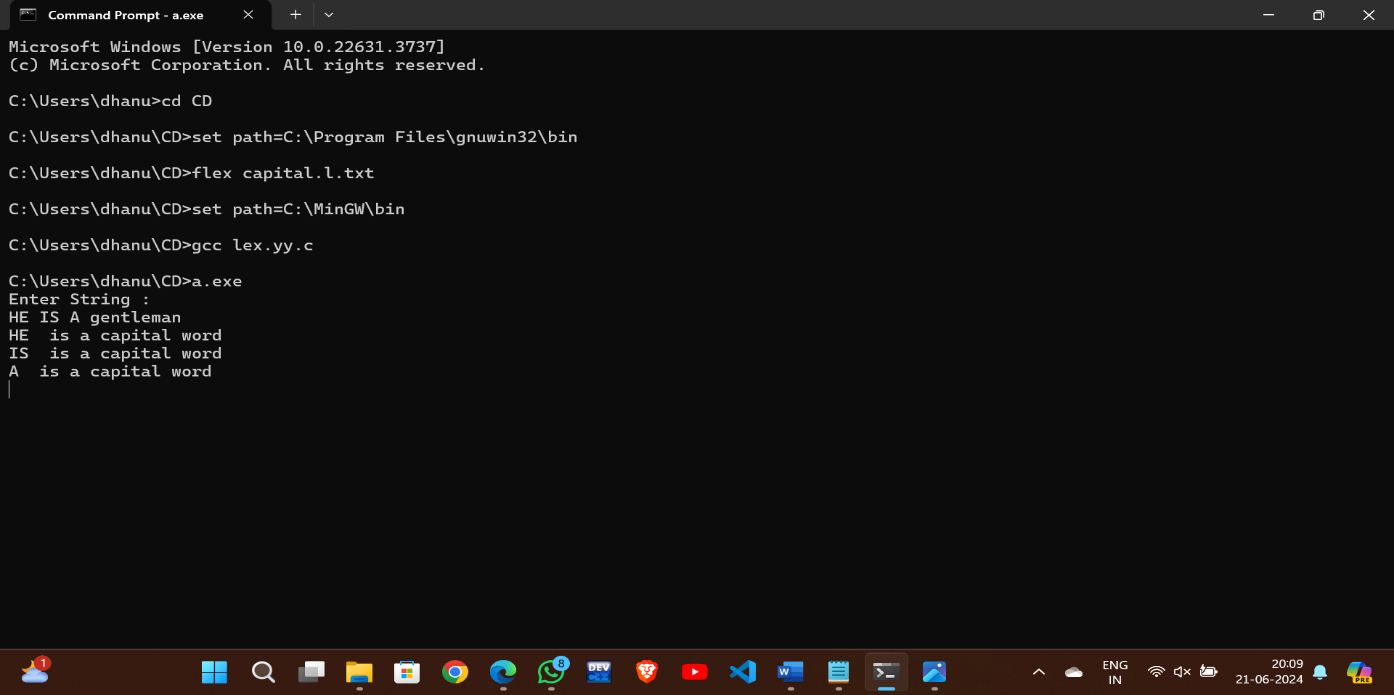
int main( )

{

printf("Enter String :\n");

yylex();

}



**EXPERIMENT-2**: LEX program to check whether the given input is digit or not.

%{

%}

%%

[0-9] { printf("Input is a digit: %s\n", yytext); }

. { printf("Input is not a digit: %s\n", yytext); }

%%

int yywrap() {

return 1;

}

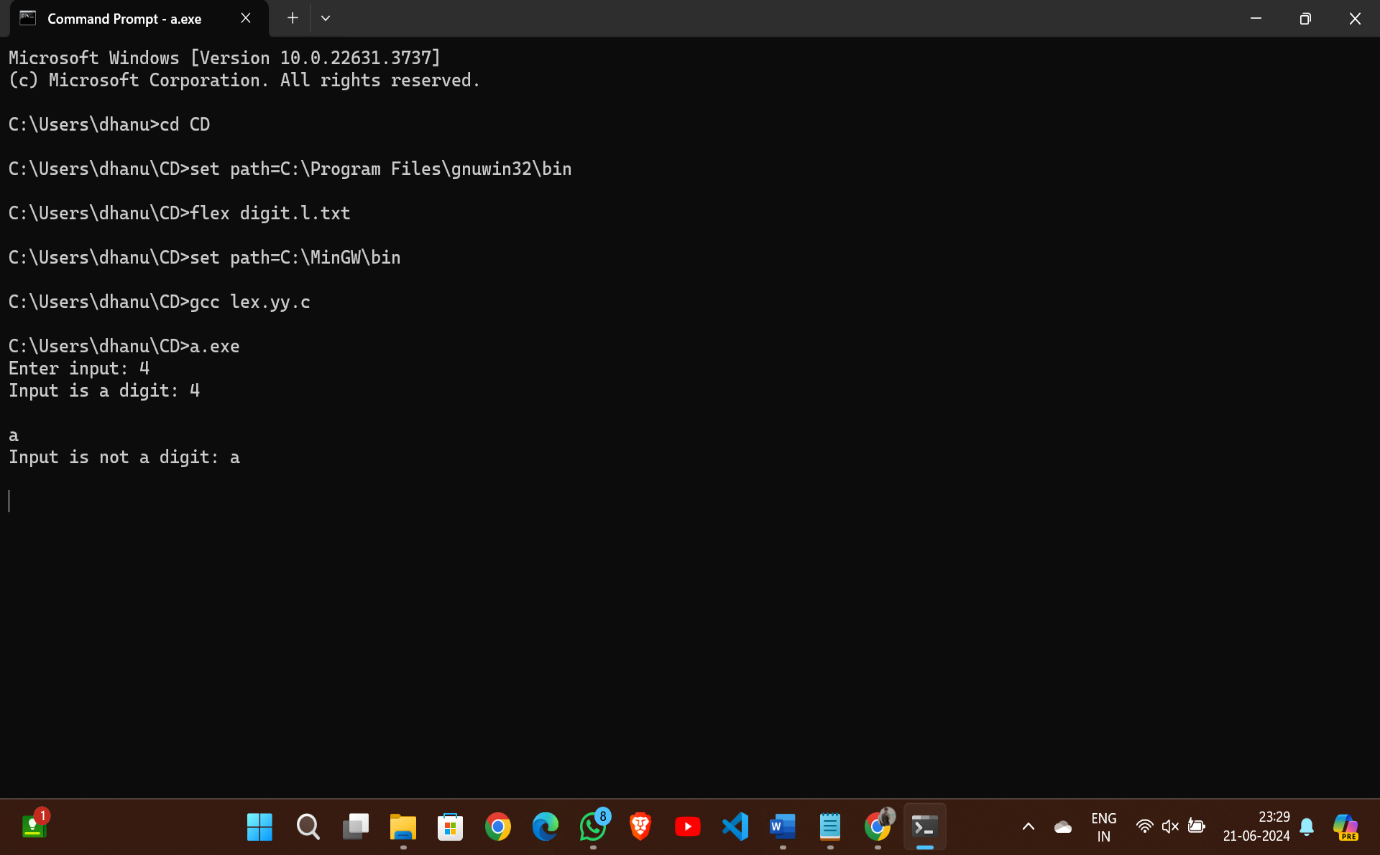
int main() {

printf("Enter input: ");

yylex();

return 0;

}



**EXPERIMENT-3**: LEX program to check whether the mobile number is valid or not.

%{

%}

%%

[6-9][0-9]{9} {printf("\nMobile Number Valid\n");}

.+ {printf("\nMobile Number Invalid\n");}

%%

int yywrap(){}

int main()

{

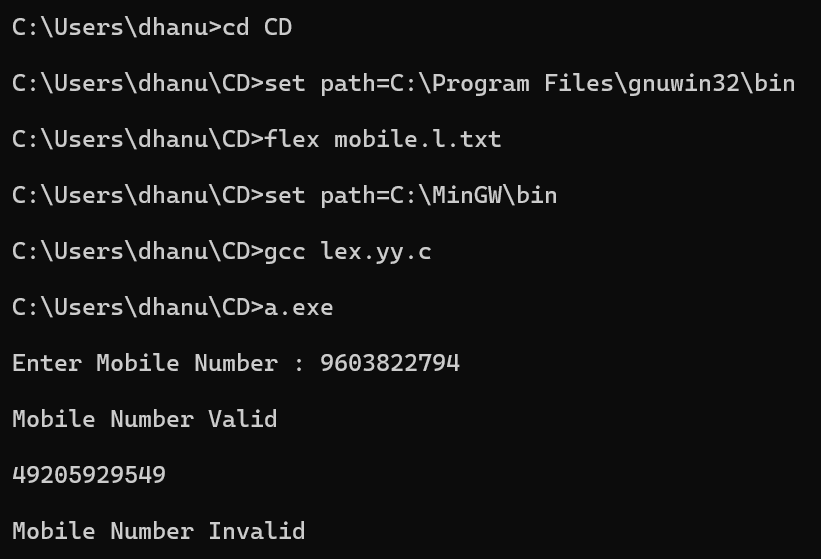
printf("\nEnter Mobile Number : ");

yylex();

printf("\n");

return 0;

}

****

**EXPERIMENT-4**: LEX program count the number of vowels and consonants in the given sentence.

%{

int vow\_count=0;

int const\_count =0;

%}

%%

[aeiouAEIOU] {vow\_count++;}

[a-zA-Z] {const\_count++;}

%%

int yywrap(){}

int main()

{

printf("Enter the string of vowels and consonants:");

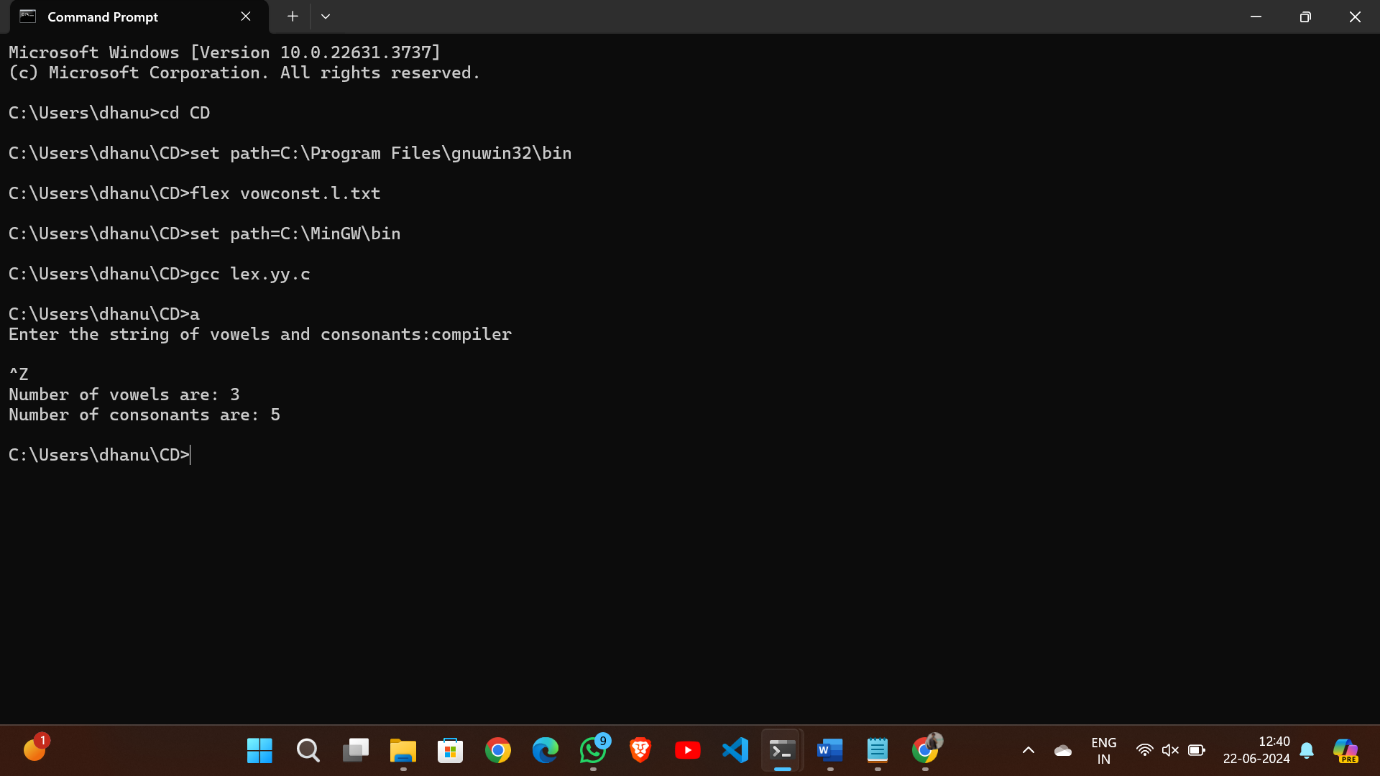
yylex();

printf("Number of vowels are: %d\n", vow\_count);

printf("Number of consonants are: %d\n", const\_count);

return 0;

}



**EXPERIMENT-5**: LEX program to separate keywords and identifiers.

%{

%}

%%

"if"|"else"|"while"|"return"|"int"|"float"|"char" { printf("Keyword: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }

[ \t\n]+

. { printf("Unknown character: %s\n", yytext); }

%%

int yywrap()

{

return 1;

}

int main()

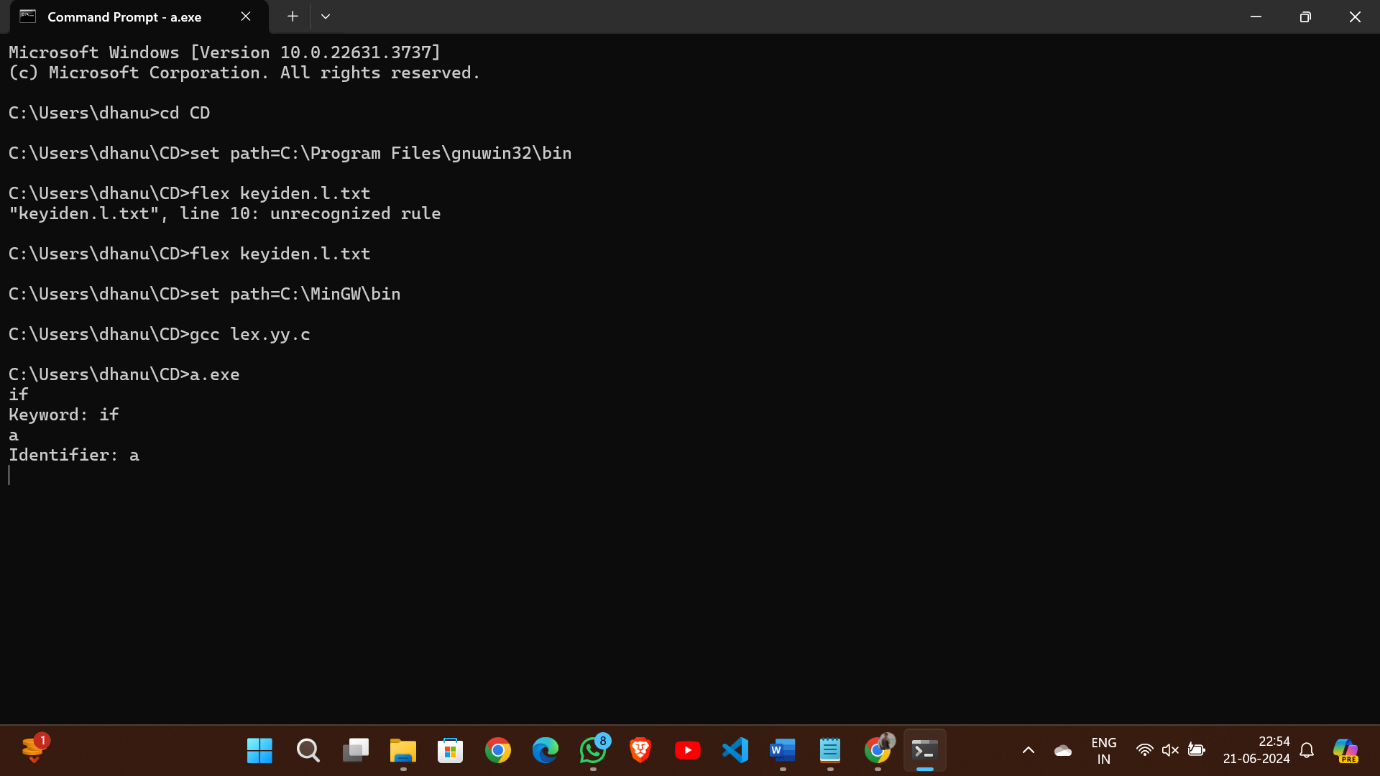
{

printf("Enter the input:");

yylex();

return 0;

}



**EXPERIMENT-6**: LEX program to identify and count positive and negative numbers.

%{

int positive\_no = 0, negative\_no = 0;

%}

%%

^[-][0-9]+ {negative\_no++; printf("negative number = %s\n", yytext);}

[0-9]+ {positive\_no++; printf("positive number = %s\n", yytext);}

%%

int yywrap(){}

int main()

{

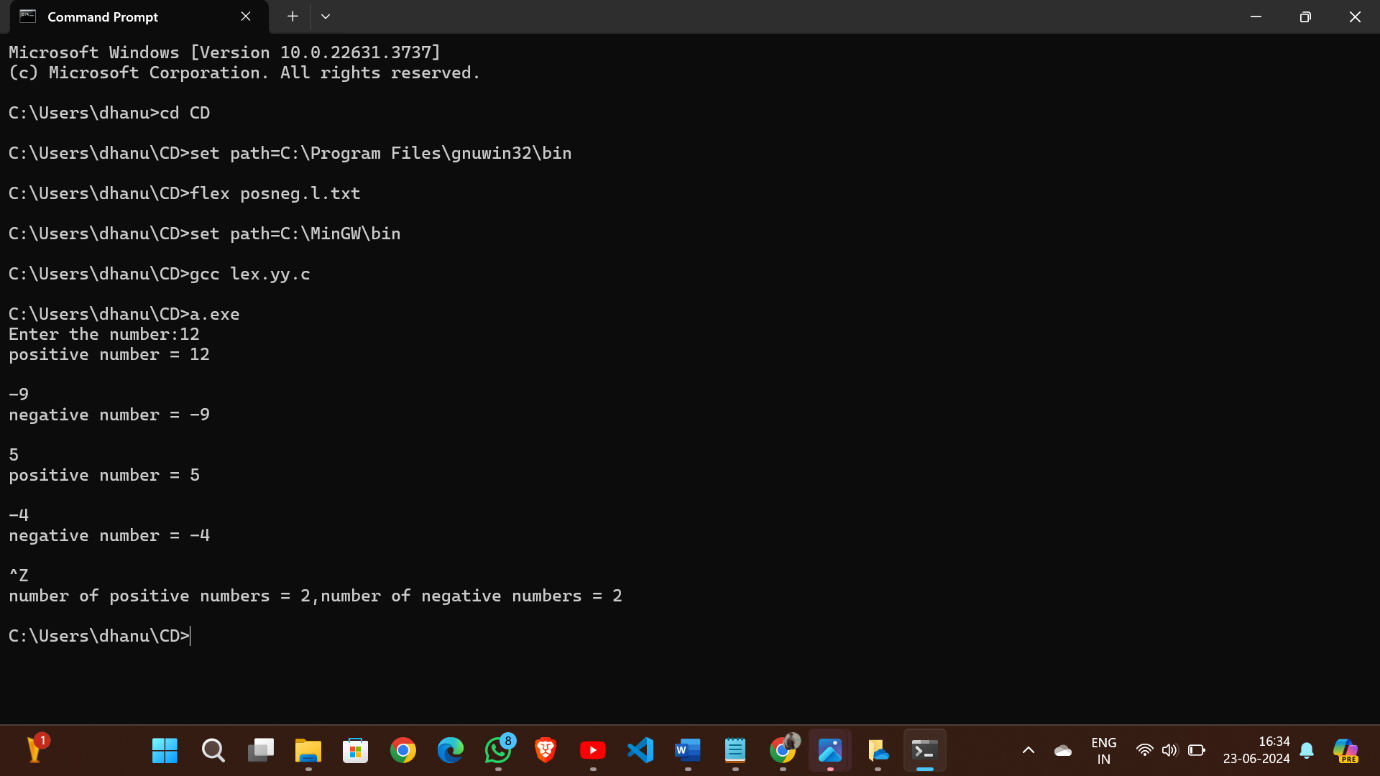
printf("Enter the number:");

yylex();

printf ("number of positive numbers = %d," "number of negative numbers = %d\n", positive\_no, negative\_no);

return 0;

}



**EXPERIMENT-7**: LEX program to recognise numbers and words in a statement.

%{

%}

%%

[0-9]+ { printf("Number: %s\n", yytext); }

[A-Za-z]+ { printf("Word: %s\n", yytext); }

[ \t\n]+

. { printf("Unknown character: %s\n", yytext); }

%%

int yywrap()

{

return 1;

}

int main()

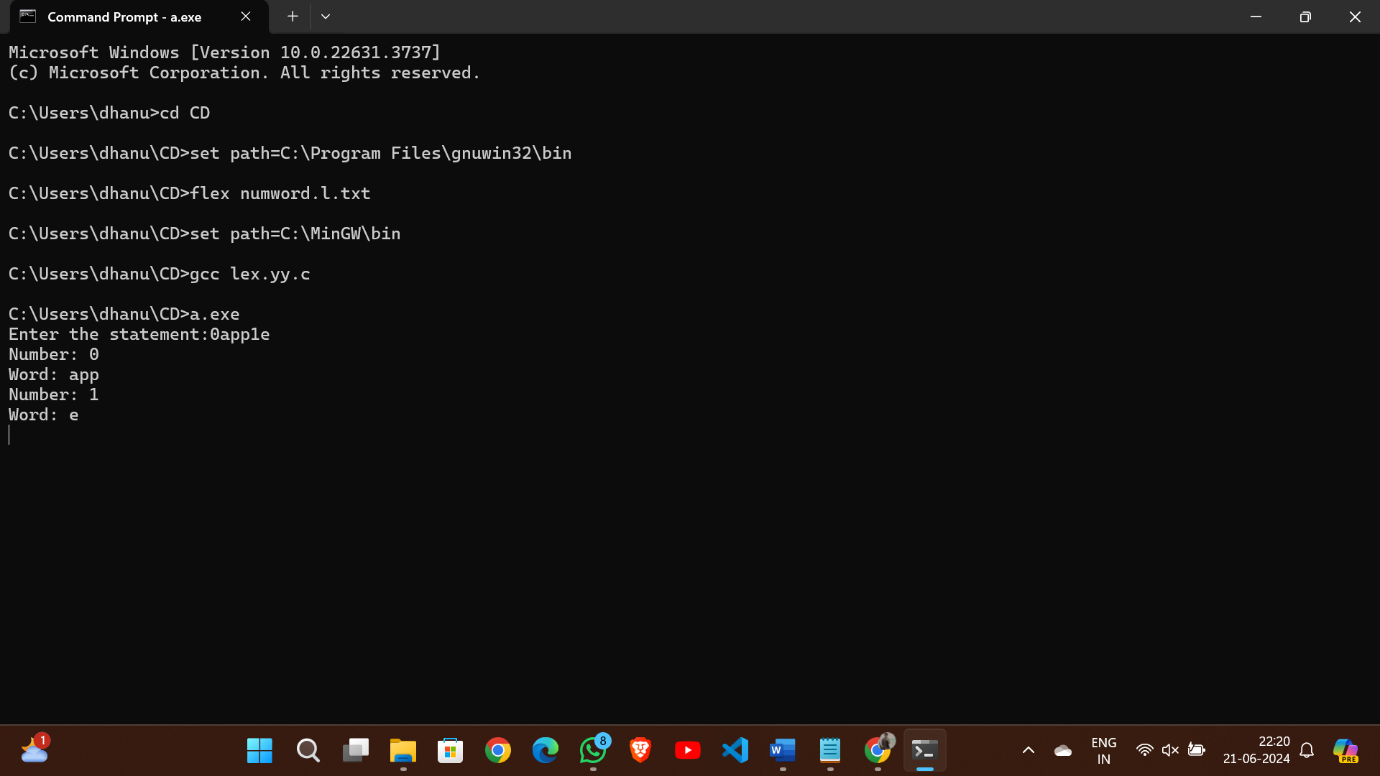
{

printf("Enter the statement:");

yylex();

return 0;

}



**EXPERIMENT-8**: LEX program to accept string starting with vowel.

%{

%}

%%

^[aeiouAEIOU].\* { printf("Accepted: %s\n", yytext); }

.\* { printf("Rejected: %s\n", yytext); }

%%

int yywrap()

{

return 1;

}

int main()

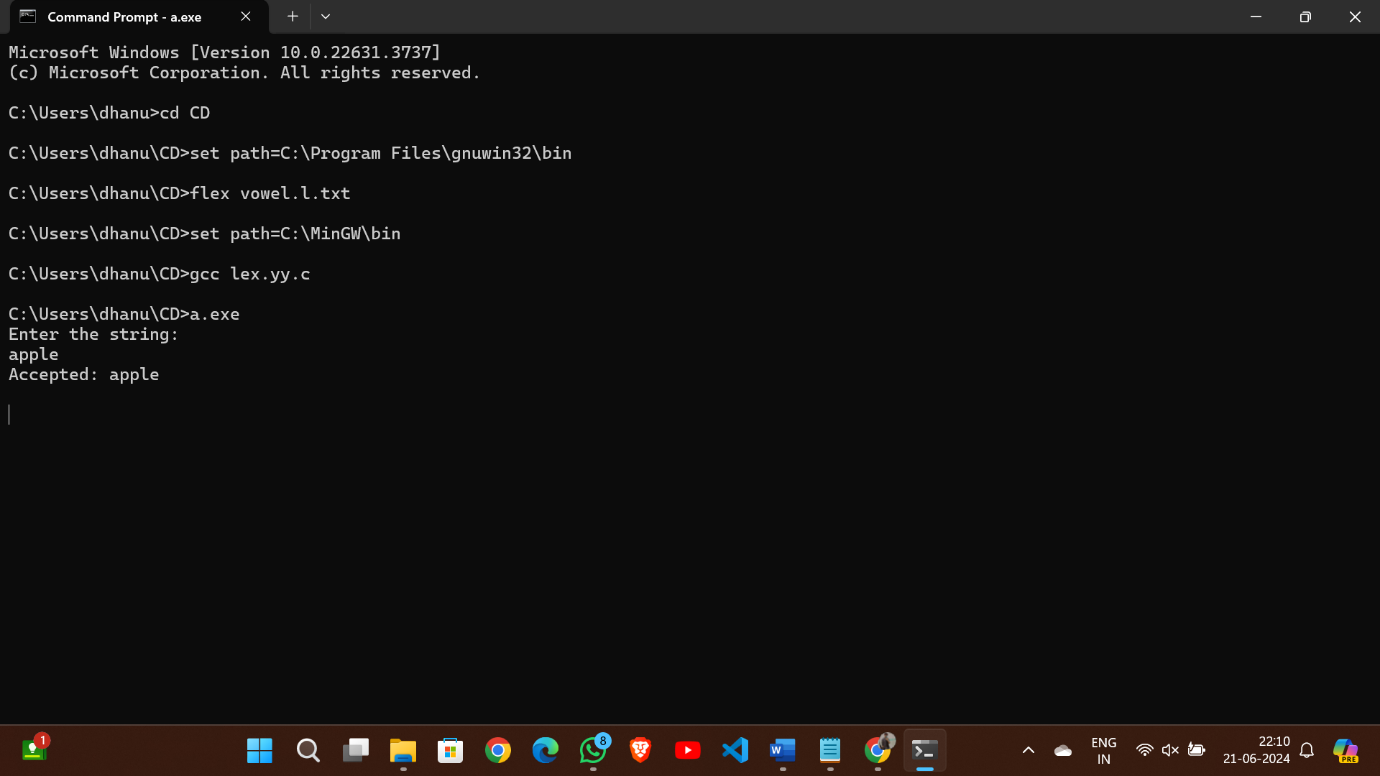
{

printf("Enter the string: \n");

yylex();

return 0;

}



**DAY 2**

**EXPERIMENT-9**: LEX program to find the length of the longest word.

%{

#include <stdio.h>

#include <string.h>

char longest\_word[256];

int longest\_length = 0;

%}

%%

[^\n\t ]+ {

int length = strlen(yytext);

if (length > longest\_length) {

longest\_length = length;

strcpy(longest\_word, yytext);

}

}

%%

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

yylex();

printf("The longest word is: %s\n", longest\_word);

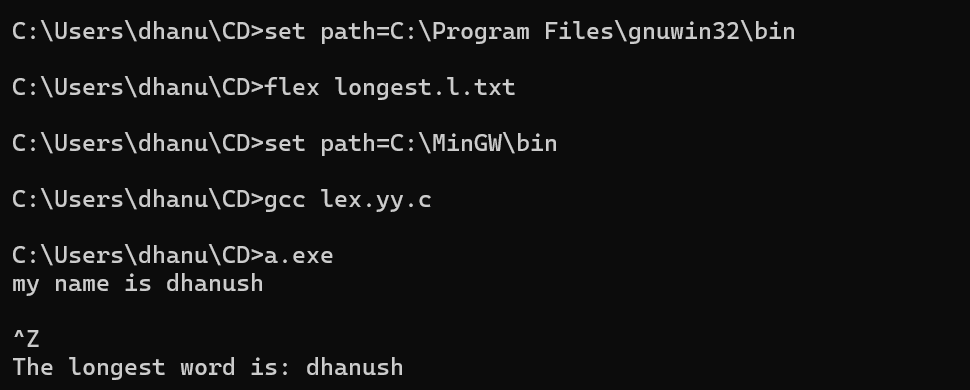
return 0;

}

int yywrap() {

return 1;

}



**EXPERIMENT-10**: LEX program to validate the URL.

%{

%}

%%

"https://www."[a-z]+".com" {printf("\nURL valid\n");}

.+ {printf("\nURL invalid\n");}

%%

int yywrap(){

return 1;

}

int main()

{

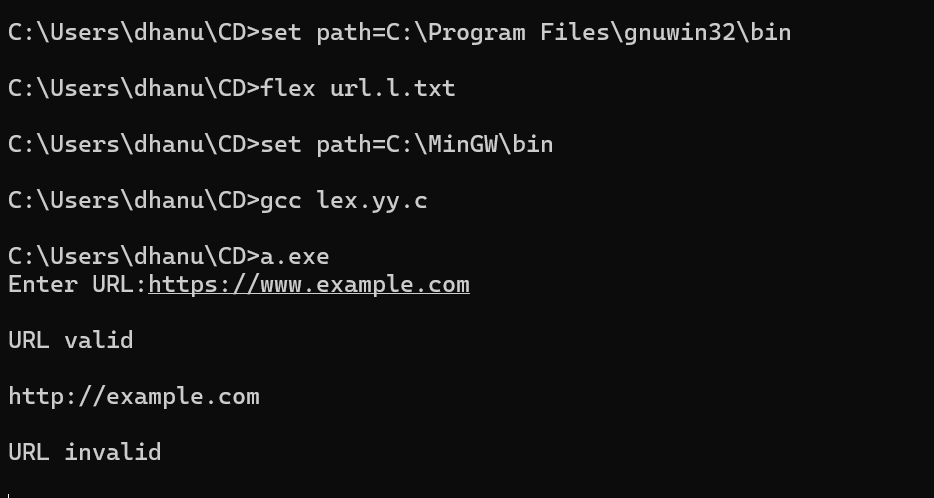
printf("Enter URL:");

yylex();

printf("\n");

return 0;

}



**EXPERIMENT-11**: LEX program to validate DOB.

%{

#include<stdio.h>

%}

%%

[0-3][0-9]\/[0-1][0-9]\/[0-9]{4} {printf("valid");}

.+ {printf("invalid");}

%%

int yywrap(){}

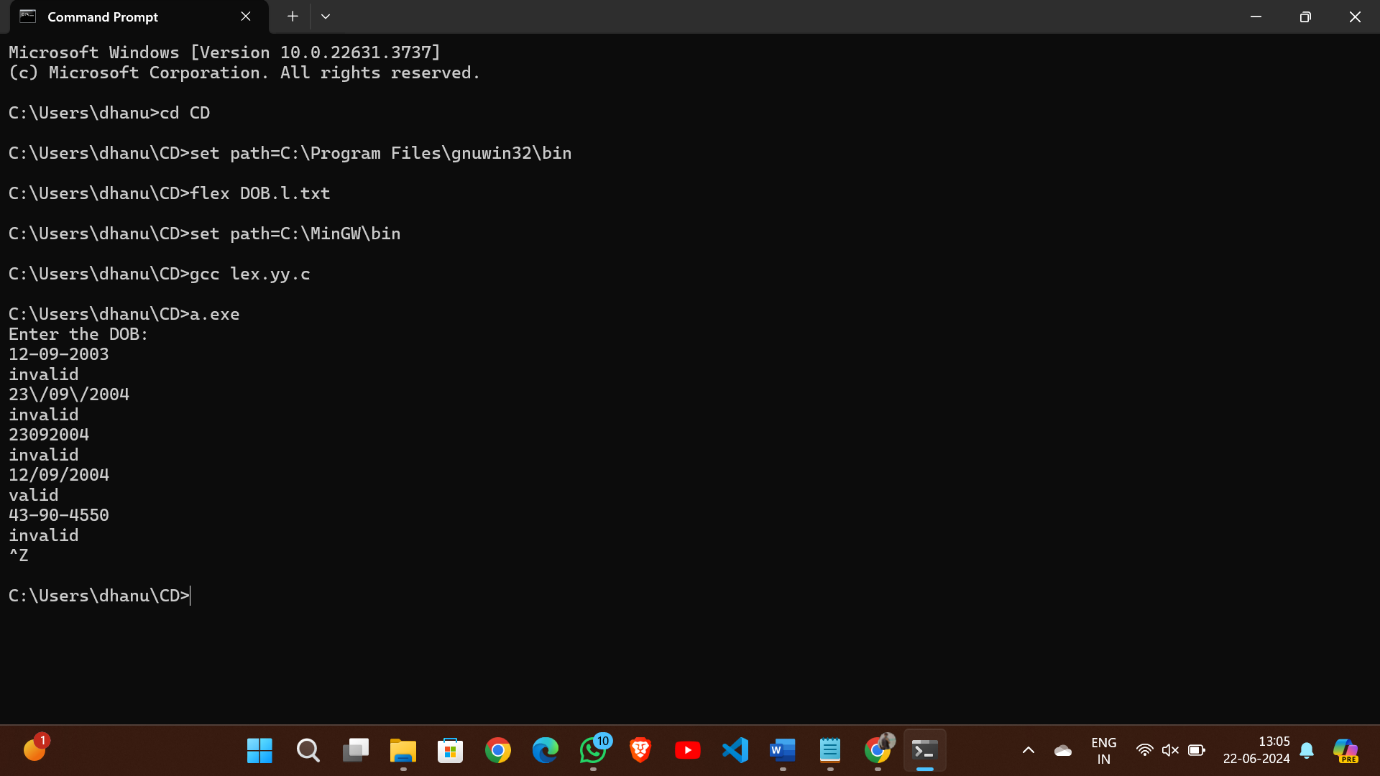
int main()

{

printf("Enter the DOB:\n");

yylex();

}



**EXPERIMENT-12**: LEX program to recognize a word and relational operator.

%{

#include <stdio.h>

%}

%%

[a-zA-Z]+ { printf("WORD: %s\n", yytext); }

"=="|"<="|">="|"!="|"<"|">" { printf("REL\_OP: %s\n", yytext); }

[ \t\n]+

. { printf("UNRECOGNIZED: %s\n", yytext); }

%%

int yywrap() { return 1;}

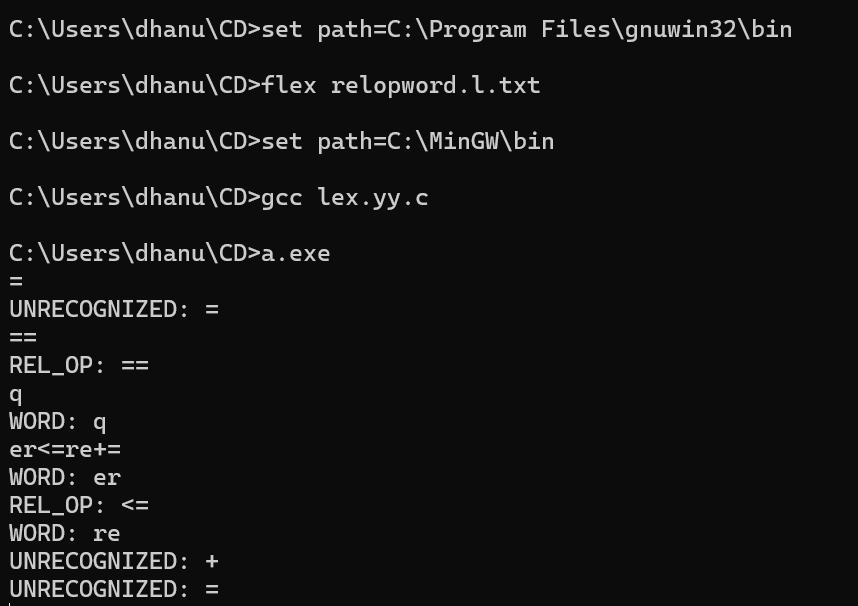
int main()

{

yylex();

return 0;

}



**EXPERIMENT-13**: LEX code to replace a word with another word in a file.

%{

#include<stdio.h>

#include<string.h>

char replace\_with [] = "hello";

char replace [] ="world";

%}

%%

[a-zA-Z]+ { if(strcmp(yytext, replace)==0)

fprintf(yyout, "%s", replace\_with);

else

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

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

%%

int yywrap()

{

return 1;

}

int main()

{

extern FILE \*yyin, \*yyout;

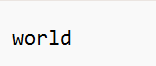
yyin=fopen("rwinput.txt", "r");

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

yylex();

}

**INPUT:** rwinput.txt:



**OUTPUT:** rwoutput.txt:

****

**EXPERIMENT-14**: LEX code to do basic mathematical operations.

%{

#include<stdio.h>

%}

%%

"="|"+"|"-"|"/"|"\*" {printf("valid");}

.+ {printf("Invalid");}

%%

int yywrap(){}

int main()

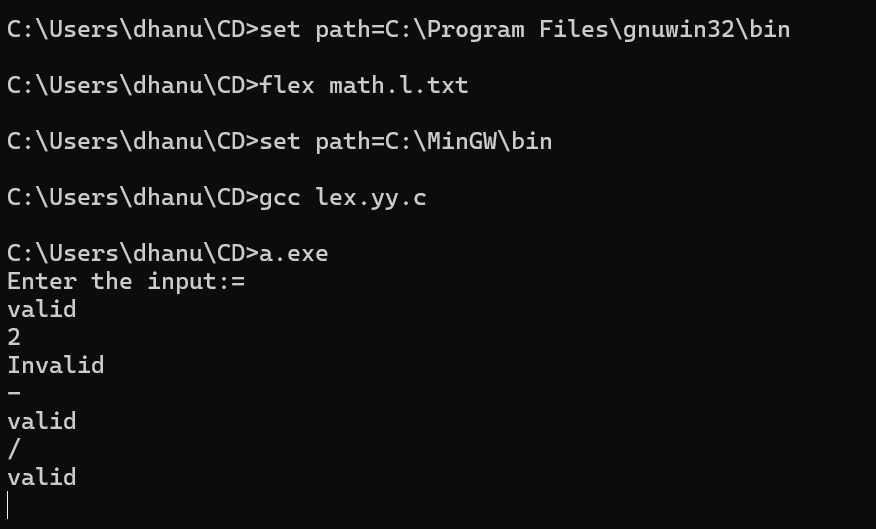
{

printf("Enter the input:");

yylex();

return 0;

}



**EXPERIMENT-15**: LEX Program to check the email address is valid or not.

%{

%}

%%

[a-z.0-9\_]+@[a-z]+".com"|".in" {printf("it is valid");}

.+ {printf("it is not valid");}

%%

int yywrap()

{}

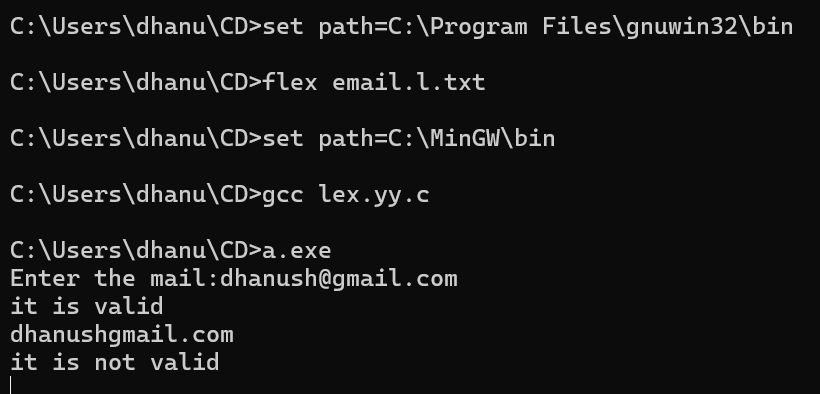
int main()

{

printf("Enter the mail:");

yylex();

}



**EXPERIMENT-16**: LEX Program to convert the substring abc to ABC from the given input string.

%{

#include <stdio.h>

%}

%%

[a-z] { printf("%c", yytext[0] - 32); }

[^a-z]+ { printf("%s", yytext); }

%%

int yywrap(void) {

return 1;

}

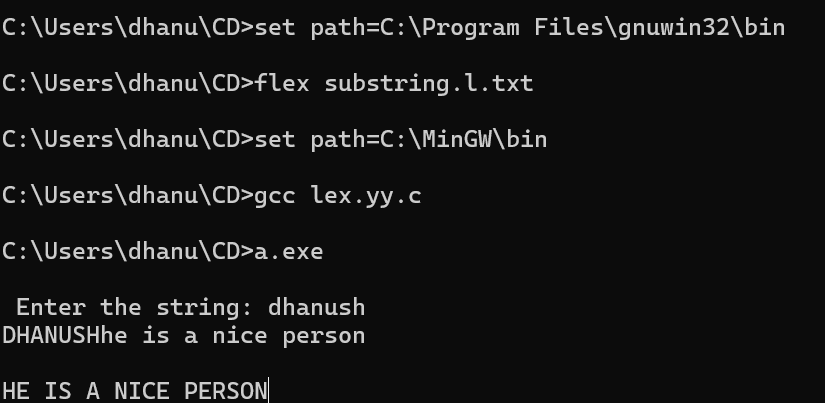
int main() {

printf("\n Enter the string: ");

yylex();

return 0;

}



**DAY 3**

**EXPERIMENT-17**: LEX specification file to take input C program from a .c file and count the number of characters, number of lines & number of words.

%{

int nchar, nword, nline;

%}

%%

\n { nline++; nchar++; }

[^ \t\n]+ { nword++, nchar += yyleng; }

. { nchar++; }

%%

int yywrap(void) {

return 1;

}

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

yyin = fopen(argv[1], "r");

yylex();

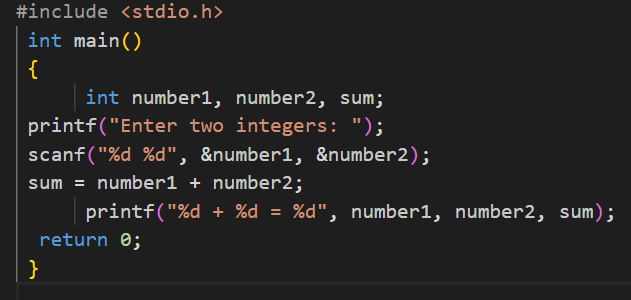
printf("Number of characters = %d\n", nchar);

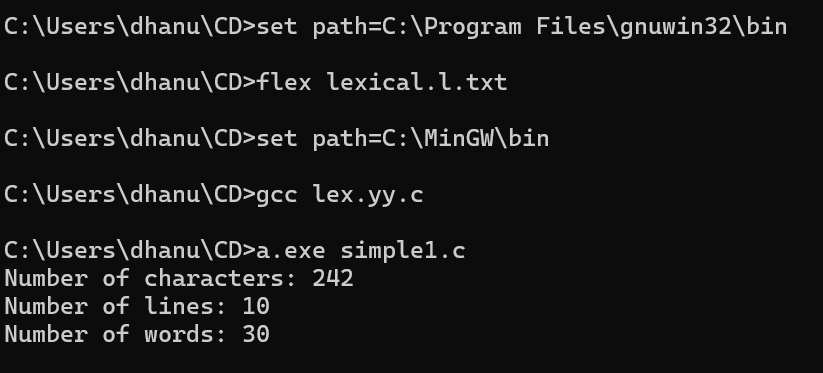
printf("Number of words = %d\n", nword);

printf("Number of lines = %d\n", nline);

fclose(yyin);

}





**EXPERIMENT-18**: LEX program to print all the constants in the given C source program file.

digit [0-9]

%{

int cons=0;

%}

%%

{digit}+ { cons++; printf("%s is a constant\n", yytext); }

.|\n { }

%%

int yywrap(void) {

return 1; }

int main(void)

{

FILE \*f;

char file[10];

printf("Enter File Name : ");

scanf("%s",file);

f = fopen(file,"r");

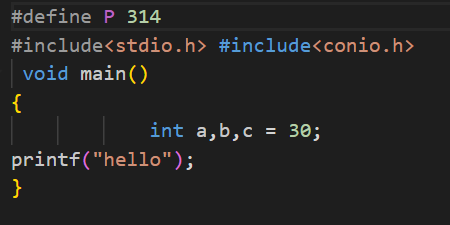
yyin = f;

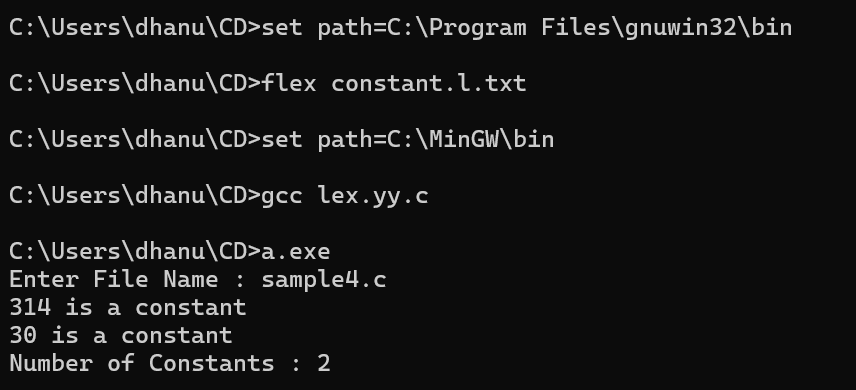
yylex();

printf("Number of Constants : %d\n", cons);

fclose(yyin);

}





**EXPERIMENT-19**: LEX program to count the number of Macros defined and header files included in the C program.

%{

int nmacro, nheader;

%}

%%

^#define { nmacro++; }

^#include { nheader++; }

.|\n { }

%%

int yywrap(void) {

return 1;

}

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

yyin = fopen(argv[1], "r");

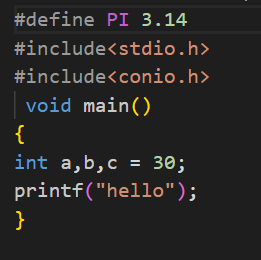
yylex();

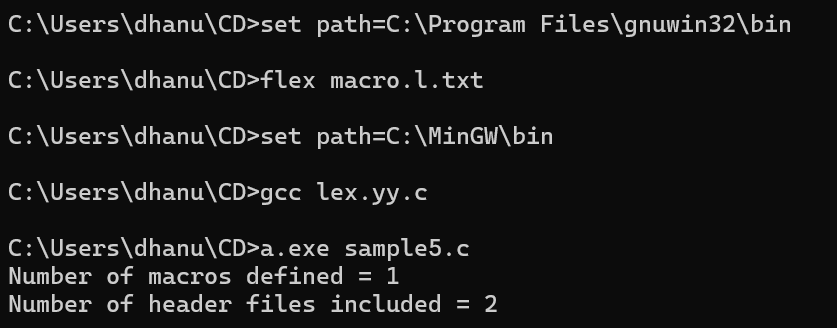
printf("Number of macros defined = %d\n", nmacro);

printf("Number of header files included = %d\n", nheader);

fclose(yyin);

}





**EXPERIMENT-20**: LEX program to print all HTML tags in the input file.

%{

int tags;

%}

%%

"<"[^>]\*> { tags++; printf("%s \n", yytext); }

.|\n { }

%%

int yywrap(void) {

return 1; }

int main(void)

{

FILE \*f;

char file[10];

printf("Enter File Name : ");

scanf("%s",file);

f = fopen(file,"r");

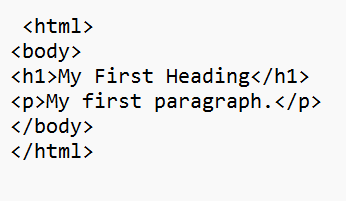
yyin = f;

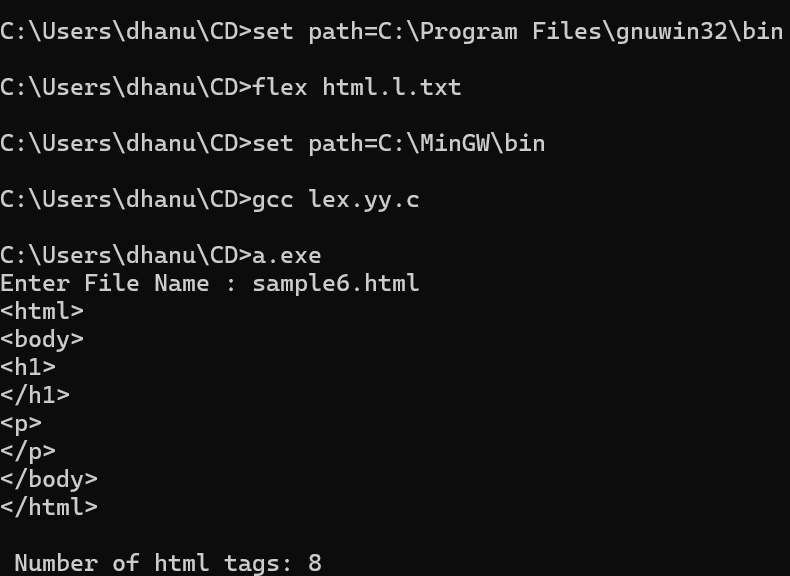
yylex();

printf("\n Number of html tags: %d",tags);

fclose(yyin);

}





**EXPERIMENT-21**: LEX program which adds line numbers to the given C program file and display the same in the standard output.

%{

int ln=0;

%}

%%

.\* {ln++; fprintf(yyout,"\n%d:%s",ln,yytext);}

%%

int yywrap(){}

int main()

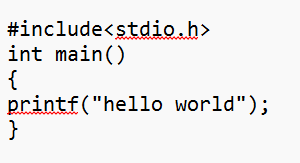
{

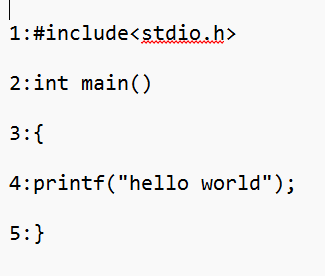
yyin=fopen("simple.txt","r");

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

yylex();

}





**EXPERIMENT-22**: LEX program to count the number of comment lines in a given C program and eliminate them and write into another file.

%{

int com=0;

%}

%s COMMENT

%%

"/\*" {BEGIN COMMENT;}

<COMMENT>"\*/" {BEGIN 0; com++;}

<COMMENT>\n {com++;}

<COMMENT>. {;}

\/\/.\* {; com++;}

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

%%

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

{

if(argc!=3)

{

printf("usage : a.exe input.c output.c\n");

exit(0);

}

yyin=fopen(argv[1],"r");

yyout=fopen(argv[2],"w");

yylex();

printf("\n number of comments are = %d\n",com);

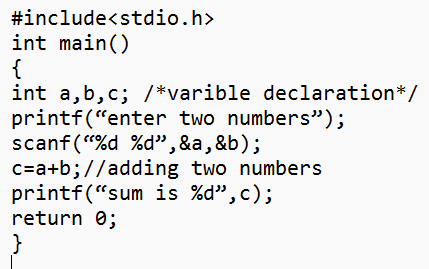
}

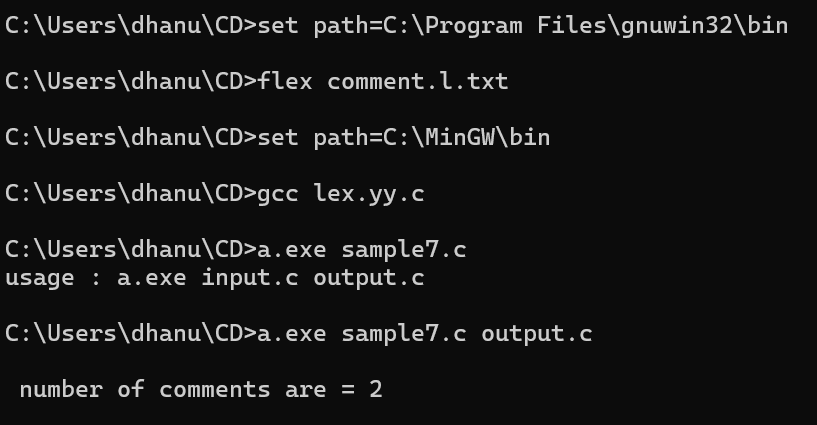
int yywrap()

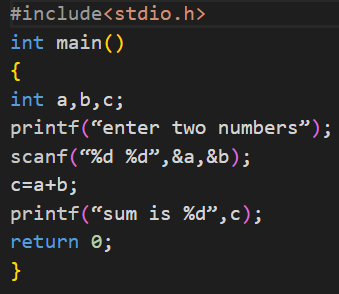
{

return 1;

}







**EXPERIMENT-23**: LEX program should separate the tokens in the given C program and display with appropriate caption.

%{

int n=0;

%}

%%

(stdio.h|conio.h) { printf("%s is a standard library\n",yytext); }

(void|main|printf|int|while|if|else|float) {n++; printf("%s is a keyword\n",yytext);}

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

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

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

"<="|"=="|"="|"+"|"-"|"\*"|"/"|"++" {n++; printf("%s is a operator\n", yytext);}

[(){}|,;] {n++;printf("%s is a separator\n", yytext);}

\"(\\.|[^"\\])\*\" {n++; printf("%s is a string literal\n", yytext); }

.|\n { }

%%

int yywrap(void)

{

return 1;

}

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

{

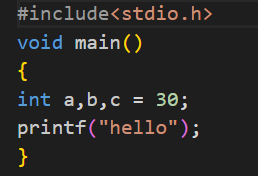
yyin = fopen(argv[1], "r");

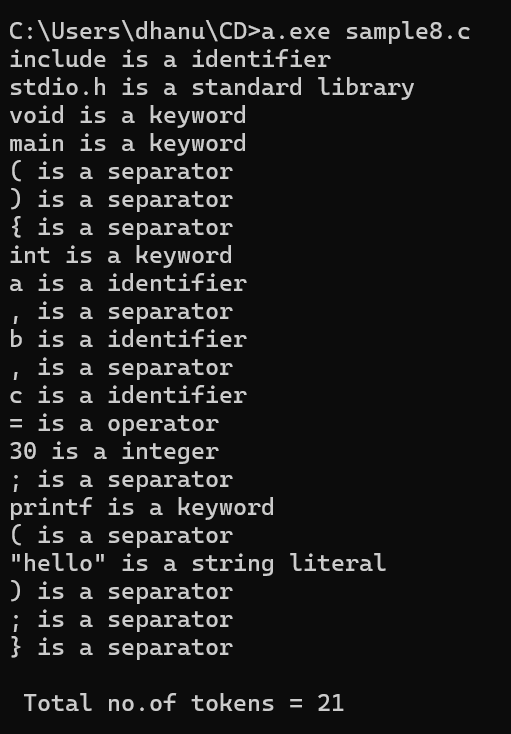
yylex();

printf("\n Total no.of tokens = %d\n", n);

fclose(yyin);

}





**EXPERIMENT-24**: LEX specification file to take input C program from a .c file and count the number of characters, number of lines & number of words.

%{

int nchar, nword, nline;

%}

%%

\n { nline++; nchar++; }

[^ \t\n]+ { nword++, nchar += yyleng; }

. { nchar++; }

%%

int yywrap(void) {

return 1;

}

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

yyin = fopen(argv[1], "r");

yylex();

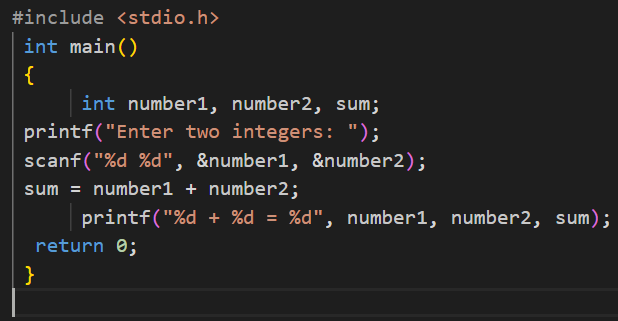
printf("Number of characters = %d\n", nchar);

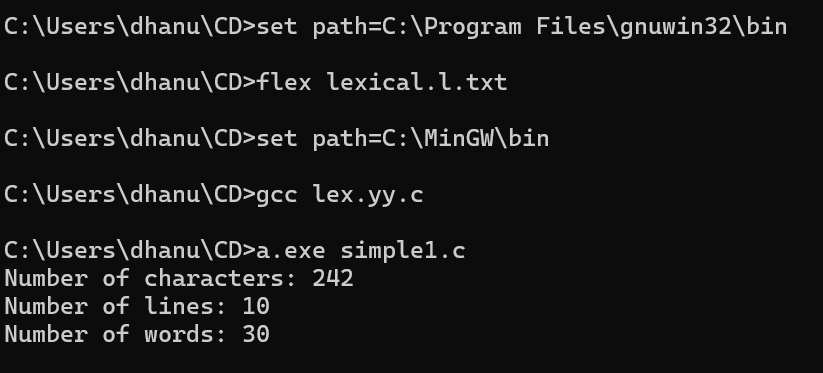
printf("Number of words = %d\n", nword);

printf("Number of lines = %d\n", nline);

fclose(yyin);

}





**DAY 4**

**EXPERIMENT-25**: LEX program to count the frequency of the given word in a given sentence.

%{

#include<stdio.h>

#include<string.h>

char word [] = "geeks";

int count = 0;

%}

%%

[a-zA-Z]+ { if(strcmp(yytext, word)==0)

count++; }

. ;

%%

int yywrap()

{

return 1;

}

int main()

{

extern FILE \*yyin, \*yyout;

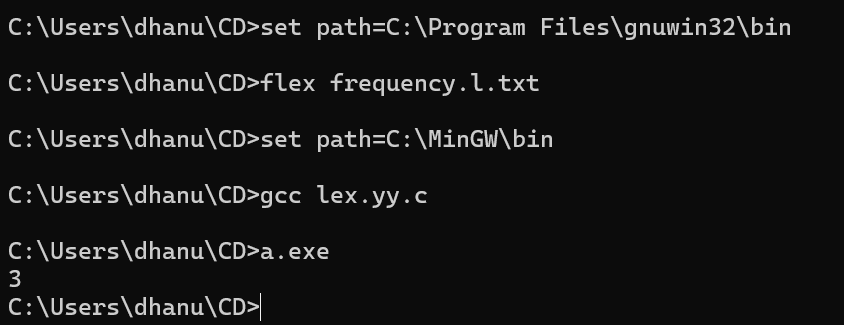
yyin=fopen("input1.txt", "r");

yylex();

printf("%d", count);

}





**EXPERIMENT-26**: Develop a lexical Analyzer to identify identifiers, constants, operators using C program.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

int isKeyword(char buffer[]){

    char keywords[32][10] = {"main","auto","break","case","char","const","continue","default",

                             "do","double","else","enum","extern","float","for","goto",

                             "if","int","long","register","return","short","signed",

                             "sizeof","static","struct","switch","typedef",

                             "unsigned","void","printf","while"};

    int i, flag = 0;

    for(i = 0; i < 32; ++i)

    {

        if(strcmp(keywords[i], buffer) == 0)

        {

            flag = 1;

            break;

        }

    }

    return flag;

}

int main()

{

    char ch, buffer[15], operators[] = "+-\*/%=(){};,";

    FILE \*fp;

    int i, j = 0;

    fp = fopen("flex\_input.txt","r");

    if(fp == NULL){

        printf("error while opening the file\n");

        exit(0);

    }

    while((ch = fgetc(fp)) != EOF){

        for(i = 0; i < sizeof(operators) - 1; ++i){

            if(ch == operators[i]){

                printf("%c is operator or punctuation\n", ch);

                break;

            }

        }

        if(isalnum(ch)){

            buffer[j++] = ch;

        }

        else if((ch == ' ' || ch == '\n' || strchr(operators, ch)) && (j != 0)){

            buffer[j] = '\0';

            j = 0;

            if(isKeyword(buffer) == 1)

                printf("%s is keyword\n", buffer);

            else

                printf("%s is identifier\n", buffer);

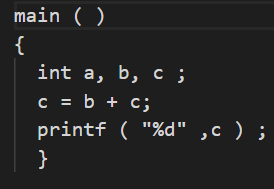
        }

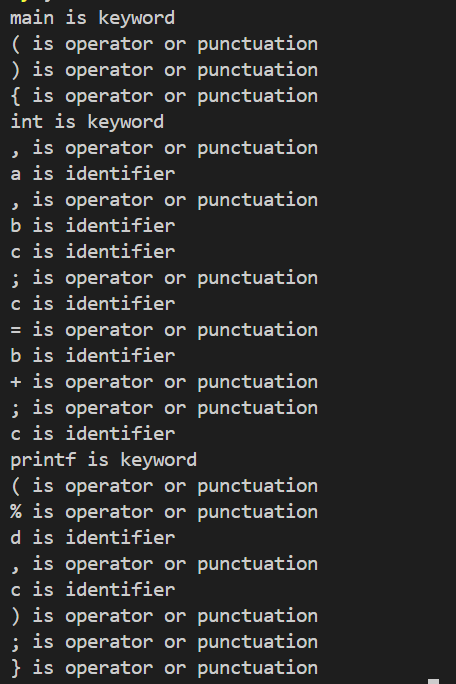
    }

    fclose(fp);

    return 0;

}





**EXPERIMENT-27**: Develop a lexical Analyzer to identify whether a given line is a comment or not.

#include<conio.h>

int main()

{

    char com[30];

    int i=2,a=0;

    printf("\n Enter comment:");

    gets(com);

    if(com[0]=='/')

    {

        if(com[1]=='/')

            printf("\n It is a comment");

        else if(com[1]=='\*')

        {

            for(i=2;i<=30;i++)

            {

                if(com[i]=='\*'&&com[i+1]=='/')

                {

                    printf("\n It is a comment");

                    a=1;

                    break;

                }

                else

                    continue;

            }

            if(a==0)

                printf("\n It is not a comment");

        }

        else

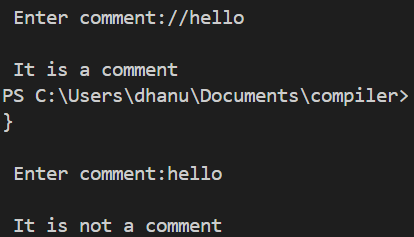
            printf("\n It is not a comment");

    }

    else

        printf("\n It is not a comment");

}



**EXPERIMENT-28**: Design a lexical Analyzer to validate operators to recognize the operators +,-,\*,/ using regular Arithmetic operators .

#include<stdio.h>

#include<conio.h>

int main()

{

    char s[5];

    printf("\n Enter any operator:");

    gets(s);

    switch(s[0])

    {

        case'>':

            if(s[1]=='=')

                printf("\n Greater than or equal");

            else

                printf("\n Greater than");

            break;

        case'<':

            if(s[1]=='=')

                printf("\n Less than or equal");

            else

                printf("\nLess than");

            break;

        case'=':

            if(s[1]=='=')

                printf("\nEqual to");

            else

                printf("\nAssignment");

            break;

        case'!':

            if(s[1]=='=')

                printf("\nNot Equal");

            else

                printf("\n Bit Not");

            break;

        case'&':

            if(s[1]=='&')

                printf("\nLogical AND");

            else

                printf("\n Bitwise AND");

            break;

        case'|':

            if(s[1]=='|')

                printf("\nLogical OR");

            else

                printf("\nBitwise OR");

            break;

        case'+':

            printf("\n Addition");

            break;

        case'-':

            printf("\nSubstraction");

            break;

        case'\*':

            printf("\nMultiplication");

            break;

        case'/':

            printf("\nDivision");

            break;

        case'%':

            printf("Modulus");

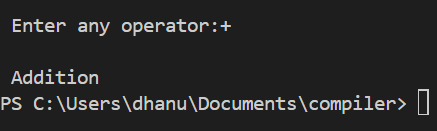
            break;

        default:

            printf("\n Not a operator");

    }

}



**EXPERIMENT-29**: Design a lexical Analyzer to find the number of whitespaces and newline characters.

#include <stdio.h>

#include <stdlib.h>

int main() {

    char ch;

    int whiteSpaces = 0, newLines = 0;

    FILE \*fp;

    // Open the file for reading

    fp = fopen("flex\_input.txt", "r");

    if (fp == NULL) {

        printf("Error while opening the file\n");

        exit(0);

    }

    // Read each character from the file

    while ((ch = fgetc(fp)) != EOF) {

        // Check for whitespace characters

        if (ch == ' ' || ch == '\t') {

            whiteSpaces++;

        }

        // Check for newline characters

        else if (ch == '\n') {

            newLines++;

        }

    }

    // Close the file

    fclose(fp);

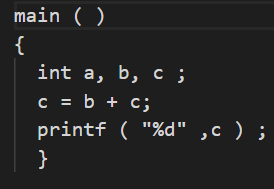
    // Print the results

    printf("Number of whitespaces: %d\n", whiteSpaces);

    printf("Number of newlines: %d\n", newLines);

    return 0;

}





**EXPERIMENT-30**: Develop a lexical Analyzer to test whether a given identifier is valid or not.

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

int main()

{

    char a[10];

    int flag, i=1;

    printf("\n Enter an identifier:");

    gets(a);

    if(isalpha(a[0]))

        flag=1;

    else

        printf("\n Not a valid identifier");

        while(a[i]!='\0')

    {

        if(!isdigit(a[i])&&!isalpha(a[i]))

        {

            flag=0;

            break;

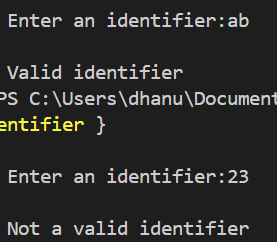
        } i++;

    }

    if(flag==1)

        printf("\n Valid identifier");

}



**EXPERIMENT-31**: Implement a C program to eliminate left recursion.

#include <stdio.h>

#include <string.h>

#define SIZE 10

int main() {

    char non\_terminal;

    char beta[SIZE], alpha[SIZE];

    int num;

    char production[10][SIZE];

    printf("Enter Number of Production: ");

    scanf("%d", &num);

    printf("Enter the grammar as E->E-A :\n");

    for (int i = 0; i < num; i++) {

        scanf("%s", production[i]);

    }

    for (int i = 0; i < num; i++) {

        printf("\nGRAMMAR : : : %s", production[i]);

        non\_terminal = production[i][0];

        int index = 3; // Reset index for each production

        if (non\_terminal == production[i][index]) {

            int alpha\_index = 0;

            int beta\_index = 0;

            index++;

            // Extract alpha

            while (production[i][index] != '\0' && production[i][index] != '|') {

                alpha[alpha\_index++] = production[i][index++];

            }

            alpha[alpha\_index] = '\0';

            // Move past '|'

            if (production[i][index] == '|') {

                index++;

            } else {

                printf(" can't be reduced\n");

                continue;

            }

            // Extract beta

            while (production[i][index] != '\0') {

                beta[beta\_index++] = production[i][index++];

            }

            beta[beta\_index] = '\0';

            printf(" is left recursive.\n");

            printf("Grammar without left recursion:\n");

            printf("%c->%s%c'\n", non\_terminal, beta, non\_terminal);

            printf("%c'->%s%c'|ε\n", non\_terminal, alpha, non\_terminal);

        } else {

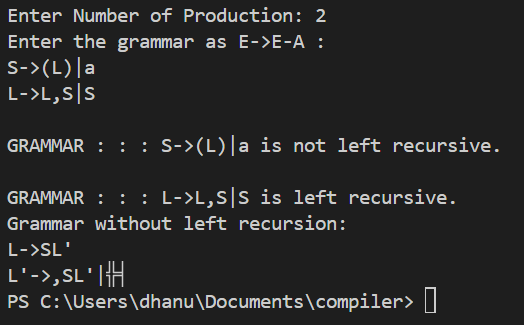
            printf(" is not left recursive.\n");

        }

    }

    return 0;

}



**EXPERIMENT-32**: Implement a C program to eliminate left factoring

#include <stdio.h>

#include <string.h>

int main() {

    char gram[100], \*parts[10], modifiedGram[20], newGram[50];

    int i, j, k = 0, l = 0, pos = 0, num\_productions = 0;

    char \*token;

    printf("Enter Productions (use '|' to separate each production): S->");

    gets(gram);

    // Split the input string by '|'

    token = strtok(gram, "|");

    while (token != NULL) {

        parts[num\_productions++] = token;

        token = strtok(NULL, "|");

    }

    // Find the longest common prefix

    for (i = 0; i < strlen(parts[0]); i++) {

        char current\_char = parts[0][i];

        int match = 1;

        for (j = 1; j < num\_productions; j++) {

            if (parts[j][i] != current\_char) {

                match = 0;

                break;

            }

        }

        if (match) {

            modifiedGram[k++] = current\_char;

            pos = i + 1;

        } else {

            break;

        }

    }

    modifiedGram[k] = 'X';

    modifiedGram[++k] = '\0';

    // Generate the new productions

    for (j = 0; j < num\_productions; j++) {

        if (j > 0) {

            newGram[l++] = '|';

        }

        for (i = pos; parts[j][i] != '\0'; i++) {

            newGram[l++] = parts[j][i];

        }

    }

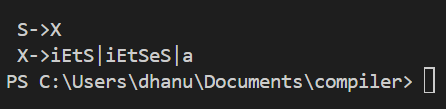
    newGram[l] = '\0';

    printf("\n S->%s", modifiedGram);

    printf("\n X->%s\n", newGram);

    return 0;

}

****

**DAY 5**

**EXPERIMENT-33**: Implement a C program to perform symbol table operations.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int cnt=0;

struct symtab

{

    char label[20];

    int addr;

}

sy[50];

void insert();

int search(char \*);

void display();

void modify();

int main()

{

int ch,val;

char lab[10];

do

{

    printf("\n1.insert\n2.display\n3.search\n4.modify\n5.exit\n");

    scanf("%d",&ch);

    switch(ch)

    {

        case 1:

            insert();

             break;

            case 2:

                display();

                break;

        case 3:

printf("enter the label");

            scanf("%s",lab);

            val=search(lab);

            if(val==1)

            printf("label is found");

            else

            printf("label is not found");

        break;

    case 4:

            modify();

        break;

    case 5:

            exit(0);

            break;

        }

    }while(ch<5);

}

void insert()

{

int val;

    char lab[10];

    int symbol;

    printf("enter the label");

    scanf("%s",lab);

    val=search(lab);

    if(val==1)

    printf("duplicate symbol");

    else

    {

        strcpy(sy[cnt].label,lab);

        printf("enter the address");

        scanf("%d",&sy[cnt].addr);

        cnt++;

    }

}

int search(char \*s)

{

    int flag=0,i; for(i=0;i<cnt;i++)

    {

        if(strcmp(sy[i].label,s)==0)

        flag=1;

    }

return flag;

}

void modify()

{

    int val,ad,i;

    char lab[10];

    printf("enter the labe:");

    scanf("%s",lab);

    val=search(lab);

    if(val==0)

    printf("no such symbol");

    else

    {

        printf("label is found \n");

        printf("enter the address");

        scanf("%d",&ad);

        for(i=0;i<cnt;i++)

        {

            if(strcmp(sy[i].label,lab)==0)

            sy[i].addr=ad;

        }

    }

}

void display()

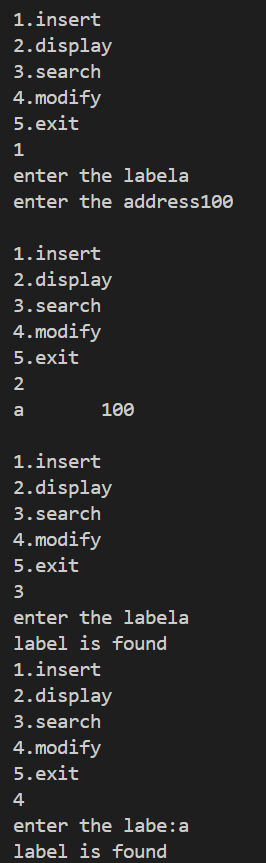
{

    int i;

    for(i=0;i<cnt;i++)

    printf("%s\t%d\n",sy[i].label,sy[i].addr);

}



**EXPERIMENT-34** implement to check whether the given input string is satisfying the grammar or not .

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main() {

    char string[50];

    int flag,count=0;

    printf("The grammar is: S->aS, S->Sb, S->ab\n");

    printf("Enter the string to be checked:\n");

    gets(string);

    if(string[0]=='a') {

        flag=0;

        for (count=1;string[count-1]!='\0';count++) {

            if(string[count]=='b') {

                flag=1;

                continue;

            } else if((flag==1)&&(string[count]=='a')) {

                printf("The string does not belong to the specified grammar");

                break;

            } else if(string[count]=='a')

            continue; else if((flag==1)&&(string[count]='\0')) {

                printf("String not accepted…..!!!!");

                break;

            } else {

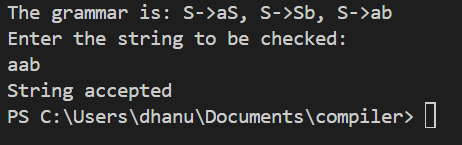
                printf("String accepted");

            }

        }

    }

}



**EXPERIMENT-35:** Write a C program to construct recursive descent parsing.

#include<stdio.h>

#include<conio.h>

#include<string.h>

char input[100];

int i,l;

void main()

{

//clrscr();

    printf("\nRecursive descent parsing for the following grammar\n");

    printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->\*FT'/@\nF->(E)/ID\n");

    printf("\nEnter the string to be checked:");

    gets(input);

    if(E())

    {

        if(input[i+1]=='\0')

            printf("\nString is accepted");

        else

            printf("\nString is not accepted");

    }

    else

        printf("\nString not accepted");

        getch();

}

E()

{

    if(T())

    {

        if(EP())

            return(1);

        else

            return(0);

    }

    else

        return(0);

}

EP()

{

    if(input[i]=='+')

    {

        i++;

        if(T())

        {

            if(EP())

                return(1);

            else

                return(0);

        }

        else

            return(0);

    }

    else

        return(1);

}

T()

{

    if(F())

    {

        if(TP())

            return(1);

        else

            return(0);

    }

    else

        return(0);

}

TP()

{

    if(input[i]=='\*')

    {

        i++;

        if(F())

        {

            if(TP())

                return(1);

            else

                return(0);

        }

        else

            return(0);

    }

    else

        return(1);

}

F()

{

    if(input[i]=='(')

    {

        i++;

        if(E())

        {

            if(input[i]==')')

            {

                i++;

                return(1);

            }

            else

                return(0);

        }

        else

            return(0);

    }

    else if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')

    {

        i++;

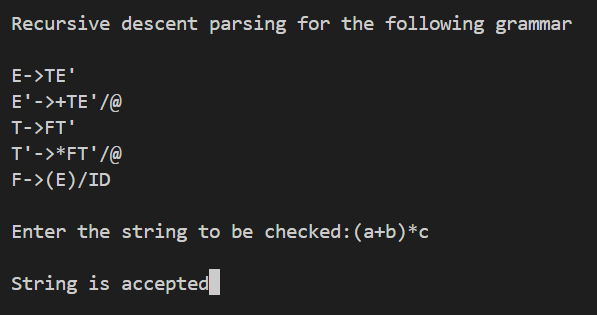
        return(1);

    }

    else

        return(0);

}



**EXPERIMENT-36:** C Program to help the students to understand about the operator precedence parsing for an expression

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

// Define precedence levels

int precedence(char operator) {

    switch (operator) {

        case '+':

        case '-':

            return 1;

        case '\*':

        case '/':

            return 2;

        case '^':

            return 3;

        default:

            return 0;

    }

}

// Define associativity of operators

int is\_right\_associative(char operator) {

    if (operator == '^')

        return 1;

    return 0;

}

// Check if character is an operator

int is\_operator(char ch) {

    return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^');

}

// Perform the actual operator precedence parsing

void operator\_precedence\_parsing(char \*expression) {

    char stack[MAX];

    int top = -1;

    char output[MAX];

    int output\_index = 0;

    for (int i = 0; i < strlen(expression); i++) {

        if (isdigit(expression[i])) {

            output[output\_index++] = expression[i];

        } else if (is\_operator(expression[i])) {

            while (top != -1 && precedence(stack[top]) >= precedence(expression[i]) && !is\_right\_associative(expression[i])) {

                output[output\_index++] = stack[top--];

            }

            stack[++top] = expression[i];

        }

    }

    while (top != -1) {

        output[output\_index++] = stack[top--];

    }

    output[output\_index] = '\0';

    printf("Parsed expression (Postfix notation): %s\n", output);

}

int main() {

    char expression[MAX];

    printf("Enter the expression: ");

    scanf("%s", expression);

    operator\_precedence\_parsing(expression);

    return 0;

}



**EXPERIMENT-37:** Write a C Program to Generate the Three address code representation for the given input statement.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

struct three

{

char data[10],temp[7];

}s[30];

int main()

{

char d1[7],d2[7]="t";

int i=0,j=1,len=0;

FILE \*f1,\*f2;

//clrscr();

f1=fopen("sum.txt","r");

f2=fopen("res.txt","w");

while(fscanf(f1,"%s",s[len].data)!=EOF)

len++;

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

strcpy(d1,"");

strcpy(d2,"t");

if(!strcmp(s[3].data,"+"))

{

fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

else if(!strcmp(s[3].data,"-"))

{

fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

for(i=4;i<len-2;i+=2)

{

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

if(!strcmp(s[i+1].data,"+"))

fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);

else if(!strcmp(s[i+1].data,"-"))

fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);

strcpy(d1,"");

strcpy(d2,"t");

j++;

}

fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);

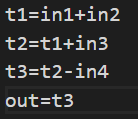
fclose(f1);

fclose(f2);

getch();

}





**EXPERIMENT-38:** Write a C program for implementing a Lexical Analyzer to Count the number of characters, words, and lines .

#include <stdio.h>

int main()

{

    char str[100];//input string with size 100

    int words=0,newline=0,characters=0; // counter variables

    scanf("%[^~]",&str);//scanf formatting

    for(int i=0;str[i]!='\0';i++)

     {

         if(str[i] == ' ')

         {

              words++;

         }

         else if(str[i] == '\n')

         {

             newline++;

              words++;//since with every next line new words start. corner case 1

         }

         else if(str[i] != ' ' && str[i] != '\n'){

         characters++;

         }

     }

    if(characters > 0)//Corner case 2,3.

    {

        words++;

        newline++;

    }

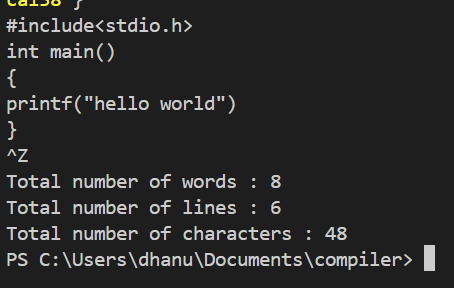
     printf("Total number of words : %d\n",words);

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

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

    return 0;

}



**EXPERIMENT-39:** Write a C Program for code optimization to eliminate common subexpression.

#include <stdio.h>

#include <string.h>

#define MAX 100

typedef struct {

    char left[MAX];

    char right[MAX];

} Expression;

void eliminate\_common\_subexpressions(Expression expressions[], int count) {

    int i, j, k;

    int optimized\_count = 0;

    Expression optimized[MAX];

    for (i = 0; i < count; i++) {

        int found = 0;

        for (j = 0; j < optimized\_count; j++) {

            if (strcmp(expressions[i].right, optimized[j].right) == 0) {

                printf("%s = %s\n", expressions[i].left, optimized[j].left);

                found = 1;

                break;

            }

        }

        if (!found) {

            optimized[optimized\_count++] = expressions[i];

            printf("%s = %s\n", expressions[i].left, expressions[i].right);

        }

    }

}

int main() {

    int count, i;

    Expression expressions[MAX];

    printf("Enter the number of expressions: ");

    scanf("%d", &count);

    getchar(); // Consume the newline character

    printf("Enter the expressions (e.g., a = b + c):\n");

    for (i = 0; i < count; i++) {

        printf("Expression %d: ", i + 1);

        fgets(expressions[i].left, MAX, stdin);

        // Split the input into left and right parts

        char \*token = strtok(expressions[i].left, "=");

        strcpy(expressions[i].left, token);

        token = strtok(NULL, "\n");

        strcpy(expressions[i].right, token);

        // Remove any leading/trailing spaces

        expressions[i].left[strcspn(expressions[i].left, "\n")] = '\0';

        expressions[i].right[strcspn(expressions[i].right, "\n")] = '\0';

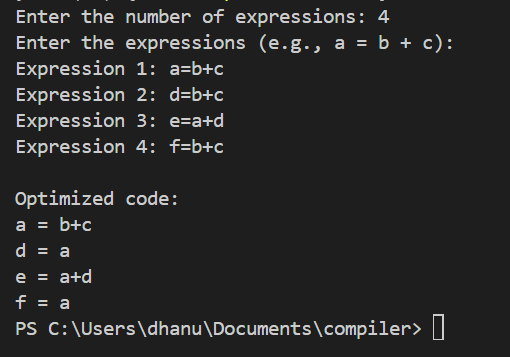
    }

    printf("\nOptimized code:\n");

    eliminate\_common\_subexpressions(expressions, count);

    return 0;

}



**EXPERIMENT-40:** Write a C program to implement the back end of the compiler.

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

    int n,i,j;

    char a[50][50];

    printf("enter the no: intermediate code:");

    scanf("%d",&n);

    for(i=0;i<n;i++)

    {

        printf("enter the 3 address code:%d:",i+1);

        for(j=0;j<6;j++)

        {

            scanf("%c",&a[i][j]);

        }

    }

    printf("the generated code is:");

    for(i=0;i<n;i++)

    {

        printf("\n mov %c,R%d",a[i][3],i);

        if(a[i][4]=='-')

        {

            printf("\n sub %c,R%d",a[i][5],i);

        }

        if(a[i][4]=='+')

        {

            printf("\n add %c,R%d",a[i][5],i);

        }

        if(a[i][4]=='\*')

        {

            printf("\n mul %c,R%d",a[i][5],i);

        }

        if(a[i][4]=='/')

        {

            printf("\n div %c,R%d",a[i][5],i);

        }

        printf("\n mov R%d,%c",i,a[i][1]);

        printf("\n");

    }

    return 0;

}

