CAPITAL WORD:

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

int main( )
{
        printf("Enter String :\n");
        yylex();
return 0;
}
int yywrap( )
{
        return 1;
}
```

EMAIL :

```
%{
    #include<stdio.h>
%}

%%
[a-zA-z0-9.]+"@gmail.com"          {printf("%s is valid email\n",yytext);}

.+                          {printf("%s is Invalid email\n",yytext);}
%%

int main()
```

```
{
        printf("\n enter the email:");
        yylex();

        return 0;
}
int yywrap()
{
   return 1;
}
```

SUBSTRING:

```
%{
#include <stdio.h>
%}

%%
"abc"    { printf("ABC"); }


%%

int main() {
   yylex();
   return 0;
```

```
}
int yywrap()
{
   return 1;
}
```

MOBILE NUMBER:

```
%{
#include<stdio.h>
%}
%%
[0-9]{10}              {printf("\nMobile Number Valid\n");}
.+                     {printf("\nMobile Number Invalid\n");}
%%
int main()
{
            printf("\nEnter Mobile Number : ");
            yylex();
            printf("\n");
            return 0;
}
int yywrap()
{
   return 1;
 }
```

SEPARATE TOKENS:

```
%{
#include <stdio.h>
%}
DIGIT      [0-9]
LETTER     [a-zA-Z]
ID         {LETTER}({LETTER}|{DIGIT})*
NUM        {DIGIT}+
WS         [ \t\n]+

%%

"include"   { printf("<INCLUDE, %s>\n", yytext); }
"void"      { printf("<VOID, %s>\n", yytext); }
"int"       { printf("<INT, %s>\n", yytext); }
"printf"    { printf("<PRINTF, %s>\n", yytext); }
"main"      { printf("<MAIN, %s>\n", yytext); }
"{"         { printf("<LEFT_BRACE, %s>\n", yytext); }
"}"         { printf("<RIGHT_BRACE, %s>\n", yytext); }
"="         { printf("<ASSIGN, %s>\n", yytext); }
","         { printf("<COMMA, %s>\n", yytext); }
";"         { printf("<SEMICOLON, %s>\n", yytext); }
"\""        { printf("<QUOTE, %s>\n", yytext); }
"("         { printf("<LEFT_PAREN, %s>\n", yytext); }
")"         { printf("<RIGHT_PAREN, %s>\n", yytext); }
{ID}        { printf("<IDENTIFIER, %s>\n", yytext); }
{NUM}       { printf("<NUMBER, %s>\n", yytext); }
{WS}        ;   // ignore whitespace

.           { printf("<UNKNOWN, %s>\n", yytext); }
```

```
%%

int main() {
   yyin = fopen("sample.c", "r");
   yylex();
   fclose(yyin);
   return 0;
}
int yywrap()
{
 return 1;
}
```

VOWELS AND CONSONANTS:

```
%{
   int vow_count=0;
   int const_count =0;
%}

%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
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;
}
int yywrap()
{
return 1;
}
```

SEPARATE KEY WORDS AND IDENTIFIERS:

```
%{
#include <stdio.h>
%}

%%

"int"|"float"|"char"|"void"    { printf("<KEYWORD, %s>\n", yytext); }
[a-zA-Z][a-zA-Z0-9]*          { printf("<IDENTIFIER, %s>\n", yytext); }
[ \t\n]                       ; // Ignore whitespace

.                             ; // Ignore other characters

%%

int main() {
    printf("Enter the C code:\n");
    yylex();
    return 0;
}
int yywrap()
```

```
{
  return 1;
}
```

POSITIVE OR NEGATIVE:

```
%{
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 main()
{
yylex();
}
int yywrap()
{
return 0;
}
```

URL FOR THE CLIENT:

```
%{
    #include <stdio.h>
%}

%%
((http)|(ftp))s?:\/\/[a-zA-Z0-9](.[a-z])+(.[a-zA-Z0-9+=?]*)* {printf("\nURL Valid\n");}

. { printf("\nURL Invalid\n"); }

%%

int main() {
   printf("\nEnter URL: ");
   yylex();
   printf("\n");
   return 0;
}

int yywrap() {
   return 1;
}
```

DOB:

```
%%

((0[1-9])|([1-2][0-9])|(3[0-1]))\/((0[1-9])|(1[0-2]))\/(19[0-9]{2}|2[0-9]{3}) printf("Valid DoB");
```

```
. printf("Invalid DoB");
%%

int main()
{
 yylex();
 return 0;
}
int yywrap()
{
  return 1;
}
```

DIGIT OR NOT:

```
%%
[0-9]+ {printf("\nValid digit \n");}
.+ printf("\nInvalid digit\n");
%%
int yywrap(){}
int main()
{
yylex();
return 0;
}
```

STARTING WITH VOWELS:

```
%{
  #include<stdio.h>

%}

%%
[aeiouAEIOU][a-zA-Z]+ {printf("\n valid\n");}
.+ {printf("invalid\n");}

%%

int main() {
   yylex();
   return 0;
}

int yywrap() {
   return 1;
}
```

FREQUENCY:

```
%{
#include<stdio.h>
#include <stdio.h>
#include <string.h>
```

```
int word_count = 0;
char target_word[100];
%}

%%
[a-zA-Z]+   {
            if (strcmp(yytext, target_word) == 0) {
               word_count++;
            }
          }
.         ;

%%

int main() {
   printf("Enter a sentence:\n");


   printf("Enter the word to count:\n");
   scanf("%s", target_word);


   yylex();

   printf("Frequency of %s: %d\n", target_word, word_count);
   return 0;
}
int yywrap()
{
 return 1;}
```

SEPARATE WORDS AND NUMBERS:

```
%{
#include <stdio.h>
%}

DIGIT      [0-9]
WORD       [a-zA-Z]+

%%

{DIGIT}+    { printf("<NUMBER, %s>\n", yytext); }
{WORD}      { printf("<WORD, %s>\n", yytext); }
[ \t\n]     ; // Ignore whitespace

.           ; // Ignore other characters

%%

int main() {
   printf("Enter the statement:\n");
   yylex();
   return 0;
}
int yywrap()
{
  return 1;
}
```

LEX CONSTANTS:

```
digit [0-9]
%{
int cons=0;
%}
%%
{digit}+ { cons++; printf("%s is a constant\n", yytext);  }
.|\n { }
%%
int yywrap() {
return 1; }
int main()
{
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);
}
```

MACRO AND HEADER:

```
%{
int nmacro, nheader;
%}
```

```
%%
^#define { nmacro++; }
^#include { nheader++; }
.|\n { }
%%
int yywrap()
{
return 1;
}
int main()
{
FILE *f;
char file[10];
printf("Enter File Name : ");
scanf("%s",file);
f = fopen(file,"r");
yyin = f;
yylex();
printf("Number of macro : %d\n", nmacro);
printf("Number of macro : %d\n", nheader);
fclose(yyin);
}
```

HTML:

```
%{
%}

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

```
. ;
%%

int yywrap(){}

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

        yyin=fopen("tags.txt","r");
        yylex();
        return 0;
}
```

STANDARD OUTPUT(20TH QUESTION)

```
%{
int line_number = 1;
%}

%%
\n  { printf("%d: %s", line_number++, yytext); }
.|\n { printf("%d: %s", line_number, yytext); }
%%

int yywrap() {
   return 1;
}

int main() {
   FILE *input_file = fopen("sample.c", "r");
```

```
    if (input_file == NULL) {
        perror("Error opening input file");
        return 1;
    }

    yyin = input_file;
    yylex();

    fclose(input_file);
    return 0;
}
```

LENGTH OF THE LONGEST WORD:

```
%{
#include <string.h>
int max_length = 0;
char longest_word[100];
%}

%%
[a-zA-Z]+ {
    if (yyleng > max_length) {
        max_length = yyleng;
        strncpy(longest_word, yytext, yyleng);
        longest_word[yyleng] = '\0';
    }
}
. ;
%%
```

```
int main() {
   printf("Enter a sentence: ");
   yylex();
   printf("Length of the longest word: %d\n", max_length);
   printf("Longest word: %s\n", longest_word);
   return 0;
}
int yywrap()
{
return 1;
}
```

NUMBER CHAR,LINRS,WORDS:

```
%{
#include<stdio.h>
int nchar, nword, nline;
%}
%%
\n { nline++, nchar+=yyleng; }
[^ \t\n]+ { nword++, nchar += yyleng; }
. { nchar++; }
%%
int main()
{
yylex();
printf("Number of characters = %d\n", nchar);
printf("Number of words = %d\n", nword);
printf("Number of lines = %d\n", nline);
return 0;
```

```
}
int yywrap()
{
return 1;
}
```

C PROGRAMMING

IDENTIFIERS,CONSTANTS,OPERATORS

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main()
{
        int i,ic=0,m,cc=0,oc=0,j;
        char b[30],op[30],id[30],con[30];
        printf("Enter the string: ");
        scanf("%[^\n]s",&b);
        for(i=0;i<strlen(b);i++)
        {
                if(isspace(b[i]))
```

```
{
        continue;
}
else if(isalpha(b[i]))
{
        id[ic]=b[i];
        ic++;
}
else if(isdigit(b[i]))
{
        m=b[i]-'0';
        i=i+1;
        while(isdigit(b[i]))
        {
                m=m*10+(b[i]-'0');
                i++;
        }
        i=i-1;
        con[cc]=m;
        cc++;
}
else
{
        if(b[i]=='*')
        {
                op[oc]='*';
                oc++;
        }
        else if(b[i]=='-')
        {
                op[oc]='-';
```

```
                        oc++;
                    }
                    else if(b[i]=='+')
                    {
                        op[oc]='+';
                        oc++;
                    }
                    else if(b[i]=='=')
                    {
                        op[oc]='=';
                        oc++;
                    }
                }
            }
            printf("Identifiers: ");
            for(j=0;j<ic;j++)
            {
                printf("%c ",id[j]);
            }
            printf("\n Constants: ");
            for(j=0;j<cc;j++)
            {
                printf("%d ",con[j]);
            }
            printf("\n Operators: ");
            for(j=0;j<oc;j++)
            {
                printf("%c ",op[j]);
            }
}
```

COMMENT OR NOT:

```
#include<stdio.h>
   #include<string.h>
   int main(){
      char com[30];
      int len;
      printf("\nEnter Comment : ");
      gets(com);
      len = strlen(com);
      if(com[0] == '/' && com[1] == '/'){
         printf("It is  Comment");
      }
      else if(com[0] == '/' && com[1] == '*' && com[len - 1] == '/' && com[len - 2] == '*' ){
         printf("It is  Comment");
      }
      else{
         printf("It is not a comment");
      }
      return 0;
   }
```

recognize the operators +,-,*,

```
#include <stdio.h>
#include <ctype.h>

int main() {
```

```
    char input[100];
    printf("Enter an arithmetic expression: ");
    fgets(input, sizeof(input), stdin);
// Tokenize the input string
    printf("Operators found: ");
    for (int i = 0; input[i] != '\0'; i++) {
        if (input[i] == '+' || input[i] == '-' || input[i] == '*' || input[i] == '/') {
            printf("%c ", input[i]);
        }
    }

    return 0;
}
```

Number of whitespaces and newline characters.

```
#include <stdio.h>
int main() {
    char ch;
    int spaceCount = 0, newlineCount = 0;

    printf("Enter text (Ctrl+D or Ctrl+Z to end input):\n");

    while (1) {
        ch = getchar();

        if (ch == EOF) {
            break;
        }
```

```
    if (ch == ' ' || ch == '\n' || ch == '\t')
                    {
        if (ch == ' ')
                            spaceCount++;
        else
                            newlineCount++;
    }
  }

  printf("\nSpaces: %d\nNewlines: %d\n", spaceCount, newlineCount);

  return 0;
}
```

output:

```
Enter a string (Ctrl+Z to end input):
Hello World!
This is a sample program.
Number of whitespaces: 5
Number of newline characters: 3
```

Identifier or Not .C

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int isValidIdentifier(const char *identifier) {
  if (!isalpha(identifier[0]) && identifier[0] != '_') {
```

```
        return 0;
    }
    for (int i = 1; i < strlen(identifier); i++) {
        if (!isalnum(identifier[i]) && identifier[i] != '_') {
            return 0;
        }
    }
    return 1;
}
            int main() {
    char identifier[30];

    printf("Enter an identifier: ");
    scanf("%s", identifier);

    if (isValidIdentifier(identifier)) {
        printf("Valid identifier\n");
    } else {
        printf("Invalid identifier\n");
    }

    return 0;
}
```

output:

Enter an identifier: myVariable123

Valid identifier

Eliminate Left Recursion .c

#include<stdio.h>

```
#include<string.h>
#define SIZE 10
 int main () {
     char non_terminal;
     char beta,alpha;
     int num,i;
     char production[10][SIZE];
     int index=3; /* starting of the string following "->" */
     printf("Enter Number of Production : ");
     scanf("%d",&num);
     printf("Enter the grammar as E->E-A :\n");
     for(i=0;i<num;i++){
         scanf("%s",production[i]);
     }
     for(i=0;i<num;i++){
         printf("\nGRAMMAR : : : %s",production[i]);
         non_terminal=production[i][0];
         if(non_terminal==production[i][index]) {
             alpha=production[i][index+1];
             printf(" is left recursive.\n");
             while(production[i][index]!=0 && production[i][index]!='|')
                 index++;
             if(production[i][index]!=0) {
                 beta=production[i][index+1];
                 printf("Grammar without left recursion:\n");
                 printf("%c->%c%c\'",non_terminal,beta,non_terminal);
                 printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
             }
             else
                 printf(" can't be reduced\n");
         }
         else
             printf(" is not left recursive.\n");
         index=3;
     }
 }
output:
Enter Number of Production : A -> a | aA | b
Enter the grammar as E->E-A :
```

Eliminate Left Factoring .c

```
#include<stdio.h>
#include<string.h>
  int main()
  {
     char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
     int i,j=0,k=0,l=0,pos;
     printf("Enter Production : S->");
     gets(gram);
     for(i=0;gram[i]!='|';i++,j++)
        part1[j]=gram[i];
     part1[j]='\0';
     for(j=++i,i=0;gram[j]!='\0';j++,i++)
        part2[i]=gram[j];
     part2[i]='\0';
     for(i=0;i<strlen(part1)||i<strlen(part2);i++)
     {
        if(part1[i]==part2[i])
        {
           modifiedGram[k]=part1[i];
           k++;
           pos=i+1;
        }
     }
     for(i=pos,j=0;part1[i]!='\0';i++,j++){
        newGram[j]=part1[i];
     }
     newGram[j++]='|';
     for(i=pos;part2[i]!='\0';i++,j++){
```

```
        newGram[j]=part2[i];
    }
    modifiedGram[k]='X';
    modifiedGram[++k]='\0';
    newGram[j]='\0';
    printf("\n S->%s",modifiedGram);
    printf("\n X->%s\n",newGram);
 }
```

output:

```
Enter Production : S->ab|ac
S->aX
 X->b|c
```

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);
}
```

output:

// Example Usage

1. Insert Operation:

Enter the label: A

Enter the address: 100

Enter the label: B

Enter the address: 200

2. Display Operation:

A 100

B 200

3. Search Operation:

Enter the label to search: A

Label is found.

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");
                        }
                }
        }
}
```

output:

The grammar is: S->aS, S->Sb, S->ab

Enter the string to be checked:

aab

String accepted

Recursive Descent Parsing.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char input[100];
int i,l;
int main()
{
```

```
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);
}
```

Operator Precedence Parsing

```
#include<stdio.h>
#include<malloc.h>
#include<string.h>

char *input;
int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};
//(E) becomes )E( when pushed to stack

int top=0,l;
char prec[9][9]={

                    /*input*/

        /*stack   +   -   *   /   ^   i   (   )   $ */

        /*  + */  '>', '>','<','<','<','<','<','>','>',
```

```
        /*  - */  '>', '>','<','<','<','<','<','>','>',

        /*  * */  '>', '>','>','>','<','<','<','>','>',

        /*  / */  '>', '>','>','>','<','<','<','>','>',

        /*  ^ */  '>', '>','>','>','<','<','<','>','>',

        /*  i */  '>', '>','>','>','>','e','e','>','>',

        /*  ( */  '<', '<','<','<','<','<','<','>','e',

        /*  ) */  '>', '>','>','>','>','e','e','>','>',

        /*  $ */  '<', '<','<','<','<','<','<','<','>',

           };

int getindex(char c)
{
switch(c)
   {
   case '+':return 0;
   case '-':return 1;
   case '*':return 2;
   case '/':return 3;
   case '^':return 4;
   case 'i':return 5;
   case '(':return 6;
   case ')':return 7;
```

```
   case '$':return 8;
   }
}

int shift()
{
stack[++top]=*(input+i++);
stack[top+1]='\0';
}



int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)//selecting handles
   {
   len=strlen(handles[i]);
   if(stack[top]==handles[i][0]&&top+1>=len)
      {
      found=1;
      for(t=0;t<len;t++)
         {
         if(stack[top-t]!=handles[i][t])
            {
            found=0;
            break;
            }
         }
      if(found==1)
         {
         stack[top-t+1]='E';
```

```
            top=top-t+1;
            strcpy(lasthandle,handles[i]);
            stack[top+1]='\0';
            return 1;//successful reduction
            }
        }
    }
return 0;
}

void dispstack()
{
int j;
for(j=0;j<=top;j++)
   printf("%c",stack[j]);
}

void dispinput()
{
int j;
for(j=i;j<l;j++)
   printf("%c",*(input+j));
}



int main()
{
int j;

input=(char*)malloc(50*sizeof(char));
```

```
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
l=strlen(input);
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
        {
        shift();
        printf("\n");
        dispstack();
        printf("\t");
        dispinput();
        printf("\tShift");
        if(prec[getindex(stack[top])][getindex(input[i])]=='>')
                {
                while(reduce())
                        {
                        printf("\n");
                        dispstack();
                        printf("\t");
                        dispinput();
                        printf("\tReduced: E->%s",lasthandle);
                        }
                }
        }

if(strcmp(stack,"$E$")==0)
   printf("\nAccepted;");
else
   printf("\nNot Accepted;");}
```

Three Address code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

void generateThreeAddressCode(char* expression) {
    char* token = strtok(expression, " ");

    int t = 1; // Temp variable counter

    while (token != NULL) {
        if (isdigit(*token) || (strlen(token) > 1 && isdigit(token[1]))) {
            printf("t%d = %s\n", t, token);
        } else if (strchr("+-*/", *token)) {
            char op = *token;
            token = strtok(NULL, " ");
            printf("t%d = t%d %c %s\n", t + 1, t, op, token);
            t++;
        }
        t++;
        token = strtok(NULL, " ");
    }
}

int main() {
    char input[] = "A + B * C - D / A"; // Replace with your input expression
```

```
    printf("Input Expression: %s\n", input);
    printf("Three-Address Code:\n");
    generateThreeAddressCode(input);

    return 0;
}
```

Count the number of characters, words, and lines .

```
#include <stdio.h>
#include <ctype.h>

int main() {
    char ch;
    int charCount = 0, wordCount = 0, lineCount = 0, inWord = 0;

    printf("Enter text (Ctrl+D or Ctrl+Z to end input on Unix or Windows):\n");

    // Read characters from the user input
    while ((ch = getchar()) != EOF) {
        charCount++;

        // Check for newline character to count lines
        if (ch == '\n') {
            lineCount++;
        }

        // Check for space, tab, or newline to determine word boundaries
        if (ch == ' ' || ch == '\t' || ch == '\n') {
            inWord = 0; // Not in a word
        } else if (!inWord) {
```

```
            inWord = 1; // Start of a new word
            wordCount++;
        }
    }

    // Output the results
    printf("\nNumber of characters: %d\n", charCount);
    printf("Number of words: %d\n", wordCount);
    printf("Number of lines: %d\n", lineCount);

    return 0;
}
```

code optimization to eliminate common subexpression

```
#include <stdio.h>

int main() {
    int a, b, c, result1, result2;

    // User input
    printf("Enter values for a, b, c: ");
    scanf("%d %d %d", &a, &b, &c);

    // Common subexpression elimination
    int common = a * b;
    result1 = common + c;
    result2 = common - c;
```

```
    // Output results
    printf("Result 1: %d\n", result1);
    printf("Result 2: %d\n", result2);

    return 0;
}
```

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;
}
```