Date:31/07/25

EXPERIMENT NO: 2

DFA IMPLEMENTATION

AIM: Write a C program to perform DFA implementation.

PROBLEM OBJECTIVE: To implement DFA using c program.

PROBLEM DEFINITION: A C program demonstrating DFA.

ALGORITHM

1. Start
2. Read the number of states.
3. Read the number of inputs.
4. Read the state..
    4.1. Check whether final state or not.
5. Read input.
    5.1. Define transition for each input.
6. Read the input string.
    6.1. Check whether the character in the string is input.
    6.2. Check the next state.
7. If it is the final state.
    7.1. Print the string is accepted.
8. Stop

PROGRAM

```
#include <stdio.h>
void main()
{
    int state[10];
    int str[10], input[10];
    char ch;
    int x[20];
    int s, n, k=0, g, j, a, i, l, t, q=0, fs, b, nxt, z;
    printf("Enter the no. of states:");
    scanf("%d", &s);
    printf("Enter the no. of ips:");
    scanf("%d", &n);
    for(i=0; i<s; i++)
    {
        printf("\nEnter the state %d:", i+1);
        scanf("%d", &state[i]);
        printf("Is final state?...y...1//...n...0:");
        scanf("%d", &a);
        if(a==1)
        {
            fs = state[i];
        }
    }
```

```
    printf("\nEnter the input:\n");
    for(i=0; i<n; i++)
    {
        scanf("%d", &input[i]);
    }

    printf("\nTransition state:");
    for(i=0; i<s; i++)
    {
        for(j=0; j<n; j++)
        {
            printf("\nq(%d,%d)=q", state[i], input[j]);
            scanf("%d", &b);
            x[k] = b;
            k++;
        }
    }

    do
    {
        printf("\nEnter the length of string:\n");
        scanf("%d", &l);
        printf("\nEnter the input string:\n");
        for(i=0; i<l; i++)
            scanf("%d", &str[i]);

        q = 0;
        for(i=0; i<l; i++)
        {
            t = 0;
            do
            {
                if(str[i] == input[t])
                {
                    nxt = x[n*q + t];
                    for(j=0; j<s; j++)
                    {
                        if(nxt == state[j])
                            q = j;
                    }
                    t++;
                }
                else
                    t++;
            } while(t != n);
        }

        if(q == fs)
            printf("String accepted .... ");
        else
            printf("String not accepted .... ");

        printf("\nDo you want to continue...If yes press 1 otherwise 0:");
        scanf("%d", &z);
    } while(z == 1);
}
```

OUTPUT

```
s7@m16:~/Desktop/shilpa$ gedit dfa.c
s7@m16:~/Desktop/shilpa$ gcc dfa.c
s7@m16:~/Desktop/shilpa$ ./a.out
Enter the no. of states:3
Enter the no. of ips:2

Enter the state 1:0
Is final state?...y...1//...n...0:0

Enter the state 2:1
Is final state?...y...1//...n...0:0

Enter the state 3:2
Is final state?...y...1//...n...0:1

Enter the input:
0
1

Transition state:
q(0,0)=q1

q(0,1)=q0

q(1,0)=q1

q(1,1)=q2

q(2,0)=q2

q(2,1)=q2

Enter the length of string:
2

Enter the input string:
0
1
String accepted.....
Do you want to continue...If yes press 1 otherwise 0:1

Enter the length of string:
3

Enter the input string:
0
0
0
String not accepted.....
Do you want to continue...If yes press 1 otherwise 0:0
s7@m16:~/Desktop/shilpa$
```

RESULT

The program has been executed successfully and output is obtained.

Date:31/07/25

EXPERIMENT NO: 3

ε - CLOSURE USING C

AIM: Write a C program to find ε-closure of all states of given NFA with epsilon transition.

PROBLEM OBJECTIVE: To find ε-closure using c program.

PROBLEM DEFINITION: A C program that demonstrates the computation of ε-closure (epsilon closure) for a given Non-deterministic Finite Automaton (NFA).

ALGORITHM

1. Start

2. Enter the number of alphabets ,number of states and transitions.

3. For i=0 to i<n of transitions ,insert the transitions to the structure defined .

4. For i=0 to i<n no. of states the transitions find the closure

5. Display the epsilon closure

6. Stop

PROGRAM

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

char result[20][20], copy[3], states[20][20];

void add_state(char a[3], int t) {
    strcpy(result[t], a);
}

void display(int n) {
    int k = 0;
    printf("\nEpsilon closure of %s = { ", copy);
    while (k < n) {
        printf("%s", result[k]);
        k++;
    }
    printf(" }\n");
}

int main()
{
    FILE *INPUT;
    INPUT = fopen("input1.txt", "r");
    char state[3];
```

```
    int end, i = 0, m, n, k = 0;
    char state1[3], input[3], state2[3];

    printf("\nEnter the no. of states: ");
    scanf("%d", &n);

    printf("\nEnter the states: \n");
    for (k = 0; k < n; k++) {
        scanf("%s", states[k]);
    }

    for (k = 0; k < n; k++) {
        int j = 0;
        strcpy(state, states[k]);
        strcpy(copy, state);
        add_state(state, j++);
        while (1) {
            end = fscanf(INPUT, "%s %s %s", state1, input, state2);
            if (end == EOF) {
                break;
            }

            if (strcmp(state, state1) == 0) {
                if (strcmp(input, "e") == 0) {
                    add_state(state2, j++);
                    strcpy(state, state2);
                }
            }
        }
        display(j);
        rewind(INPUT);
    }

    return 0;
}
```

input1.txt

```
q0 e q1
q0 1 q2
q1 e q2
```

OUTPUT

```
s7@m16:~/Desktop/shilpa$ gedit enfa.c
s7@m16:~/Desktop/shilpa$ gedit input.txt
s7@m16:~/Desktop/shilpa$ gcc enfa.c
s7@m16:~/Desktop/shilpa$ ./a.out

Enter the no. of states: 3

Enter the states:
q0
q1
q2

Epsilon closure of q0 = { q0q1q2 }

Epsilon closure of q1 = { q1q2 }

Epsilon closure of q2 = { q2 }
s7@m16:~/Desktop/shilpa$
```

RESULT

The program has been executed successfully and output is obtained.

Date:07/08/25

EXPERIMENT NO: 4

RECURSIVE DESCENT PARSING

AIM: Write a C program to perform recursive descent parsing for the following grammer.

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

PROBLEM OBJECTIVE: To implement recursive descent parsing using c program.

ALGORITHM

1. Start
2. Define the variables.
3. Read the arithmetic expression.
4. E()
    - 4.1.T()
    - 4.2.EPrime()
        - 4.2.1.If input[i] = + then
            - 4.2.1.1.i =i+1
            - 4.2.1.2.T()
            - 4.2.1.3.EPrime()
5. T()
    - 5.1.F()
    - 5.2.TPrime()
6. TPrime()
    - 6.1.If input[i] = * then
        - 6.1.1.i =i+1
        - 6.1.2.F()
        - 6.1.3.TPrime()
7. F()
    - 7.1. If input[i] = (then
        - 7.1.1.i =i+1
        - 7.1.2.E()
        - 7.1.3. If input[i] =) then set i =i+1
    - 7.2.If isalpha(input([i]) then set i=i+1
    - 7.3.Else set error=1
8. If length of input string=I and error=0 then print accepted.
9. Else print rejected.
10. Stop.

PROGRAM

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

char input[10];
int i = 0, error = 0;
```

```
void E();
void T();
void Eprime();
void Tprime();
void F();

int main() {
   printf("Enter an arithmetic expression: ");
   scanf("%s", input);

   E();

   if ((strlen(input) == i) && error == 0)
      printf("\nAccepted\n");
   else
      printf("Rejected\n");

   return 0;
}

void E() {
   T();
   Eprime();
}

void Eprime() {
   if (input[i] == '+') {
      i++;
      T();
      Eprime();
   }
}

void T() {
   F();
   Tprime();
}

void Tprime() {
   if (input[i] == '*') {
      i++;
      F();
      Tprime();
   }
}

void F() {
   if (input[i] == '(') {
      i++;
      E();
```

```
        if (input[i] == ')')
            i++;
        else
            error = 1;
    }
    else if (isalpha(input[i]) || input[i] == '_') {
        i++;
        while (isalnum(input[i]) || input[i] == '_')
            i++;
    }
    else
        error = 1;
}
```

OUTPUT

RESULT

The program has been executed successfully and output is obtained.

Date:07/08/25

EXPERIMENT NO: 5

SHIFT REDUCE PARSING

AIM: Write a C program to perform shift reduce parsing.

PROBLEM OBJECTIVE: To implement shift reduce parsing using c program.

ALGORITHM

1. Start
2. Display grammar rules.
3. Read the input string.
4. Intialize
   4.1 Stack ← empty
   4.2 Action ← SHIFT
5. While input is not empty, repeat
   5.1 SHIFT: Move next input symbol to stack and mark it as read.
   5.2 Print stack, input, and action.
   5.3 REDUCE (check stack for patterns):
      5.3.1 If i → replace with E
      5.3.2 If E + E → replace with E
      5.3.3 If E * E → replace with E
      5.3.4 If (E) → replace with E
   5.4 Print stack and input after reduction.
6. After input is fully read, perform reductions again if possible
7. If final stack = E then
      Print "Accepted"
   Else
      Print "Rejected"
8. Stop

PROGRAM

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

int z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];

void check() {
   strcpy(ac, "REDUCE TO E->");

   for (z = 0; z < c; z++) {
      if (stk[z] == 'i') {
```

```
            printf("%si", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
        }
    }

    for (z = 0; z < c - 2; z++) {
        if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E') {
            printf("%sE+E", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }

    for (z = 0; z < c - 2; z++) {
        if (stk[z] == 'E' && stk[z + 1] == '*' && stk[z + 2] == 'E') {
            printf("%sE*E", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }

    for (z = 0; z < c - 2; z++) {
        if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')') {
            printf("%s(E)", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    return;
}

int main() {
    printf("GRAMMAR is \nE->E+E \nE->E*E \nE->(E) \nE->i");
    printf("\nEnter the string: ");
    scanf("%s", a);
    c = strlen(a);
    strcpy(act, "SHIFT");
    printf("\nstack \t input \t action");
    printf("\n$\t%s$\t",a);
```

```
    for (i = 0; j < c; i++, j++) {
        printf("%s", act);
        stk[i] = a[j];
        stk[i + 1] = '\0';
        a[j] = ' ';
        printf("\n$%s\t%s$\t", stk, a);
        check();
}
    check();

if(stk[0] == 'E' && stk[1] == '\0')
    printf("Accepted\n");
else
    printf("Rejected\n");
}
```

OUTPUT

```
s7@m16:~/Desktop/shilpa$ gedit srp.c
s7@m16:~/Desktop/shilpa$ gcc srp.c
s7@m16:~/Desktop/shilpa$ ./a.out
GRAMMAR is
E->E+E
E->E*E
E->(E)
E->i
Enter the string: (i+i)*i

stack    input   action
$       (i+i)*i$        SHIFT
$(       i+i)*i$        SHIFT
$(i       +i)*i$        REDUCE TO E->i
$(E       +i)*i$        SHIFT
$(E+       i)*i$        SHIFT
$(E+i       )*i$        REDUCE TO E->i
$(E+E       )*i$        REDUCE TO E->E+E
$(E         )*i$        SHIFT
$(E)         *i$        REDUCE TO E->(E)
$E           *i$        SHIFT
$E*           i$        SHIFT
$E*i           $        REDUCE TO E->i
$E*E           $        REDUCE TO E->E*E
$E             $        Accepted
s7@m16:~/Desktop/shilpa$ ./a.out
GRAMMAR is
E->E+E
E->E*E
E->(E)
E->i
Enter the string: i-i*i

stack    input   action
$       i-i*i$  SHIFT
$i       -i*i$  REDUCE TO E->i
$E       -i*i$  SHIFT
$E-       i*i$  SHIFT
$E-i       *i$  REDUCE TO E->i
$E-E       *i$  SHIFT
$E-E*       i$  SHIFT
$E-E*i       $  REDUCE TO E->i
$E-E*E       $  REDUCE TO E->E*E
$E-E         $  Rejected
s7@m16:~/Desktop/shilpa$
```

RESULT

The program has been executed successfully and output is obtained.

Date:12/09/25

EXPERIMENT NO: 7

CONSTANT PROPAGATION

AIM: Write a C program to perform constant propagation.

PROBLEM OBJECTIVE: To implement constant propagation in C for code optimization.

PROBLEM DEFINITION: A C program that performs constant propagation by replacing variables with constant values to simplify and optimize code.

ALGORITHM

1. Start
2. Read the no. of expressions, *n*.
3. For each expression (from i = 0 to n–1):
   3.1 Read operator op[i], operand1 op1[i], operand2 op2[i] and result res[i].
   3.2 Set flag[i] = 0.
4. For each expression (from i = 0 to n–1):
   4.1 If both operands are constants:
   4.1.1 Convert op1[i] and op2[i] to integers.
   4.1.2 Perform the operation (+, –, *, /) based on op[i].
   4.1.3 Store the result as a string in res[i].
   4.1.4 Set flag[i] = 1.
   4.1.5 Set change = true (to propagate the result).
5. If change = true:
   5.1 For each expression (from i = 0 to n–1):
   5.1.1 Replace op1[i] or op2[i] with res[j] if it matches res[i].
6. For each expression (from i = 0 to n–1):
   6.1 If flag[i] == 0, print the original expression.
7. Stop

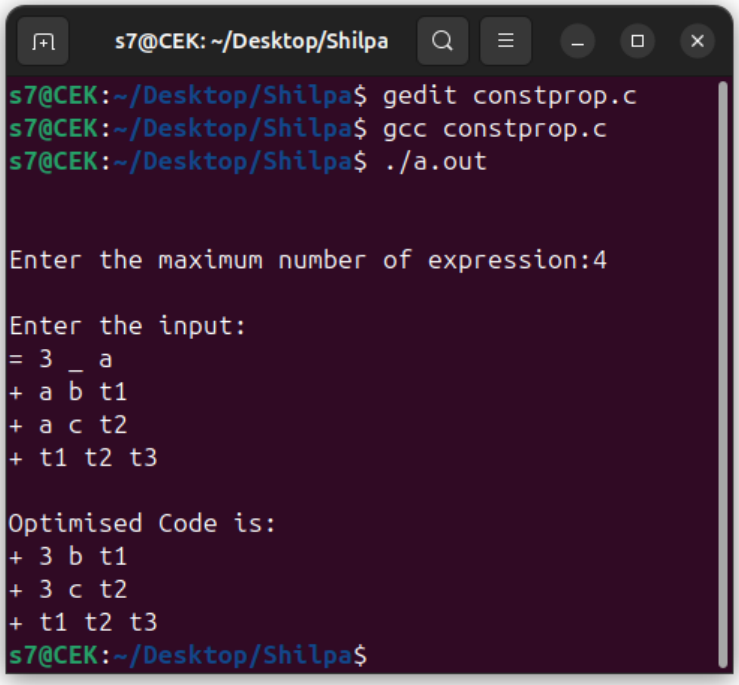
PROGRAM

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

void input();
void output();
void constant();
void change(int p,char *res);

struct expr {
   char op[2], op1[5], op2[5], res[5];
   int flag;
} arr[10];
```

```
int n;

void main() {
   input();
   constant();
   output();
}

void input() {
   int i;
   printf("\n\nEnter the maximum number of expression:");
   scanf("%d", &n);
   printf("\nEnter the input:\n");

   for (i = 0; i < n; i++) {
      scanf("%s", arr[i].op);
      scanf("%s", arr[i].op1);
      scanf("%s", arr[i].op2);
      scanf("%s", arr[i].res);
      arr[i].flag = 0;
   }
}

void constant() {
   int i;
   int op1, op2, res;
   char op, res1[5];

   for (i = 0; i < n; i++) {
      if (isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=") == 0) {
         op1 = atoi(arr[i].op1);
         op2 = atoi(arr[i].op2);
         op  = arr[1].op[0];

         switch (op) {
            case '+': res = op1 + op2; break;
            case '-': res = op1 - op2; break;
            case '*': res = op1 * op2; break;
            case '/': res - op1 / op2; break;
            case '=': res = op1; break;
         }

         sprintf(res1, "%d", res);
         arr[i].flag = 1;
         change(i, res1);
      }
   }
}

void output() {
```

```
    int i = 0;
    printf("\nOptimised Code is:");

    for (i = 0; i < n; i++) {
        if (!arr[i].flag) {
            printf("\n%s %s %s %s", arr[i].op, arr[i].op1, arr[i].op2, arr[i].res);
        }
    }
    printf("\n");
}

void change(int p, char *res) {
    int i;
    for (i = p + 1; i < n; i++) {
        if (strcmp(arr[p].res, arr[i].op1) == 0)
            strcpy(arr[i].op1, res);
        else if (strcmp(arr[p].res, arr[i].op2) == 0)
            strcpy(arr[i].op2, res);
    }
}
```

OUTPUT



RESULT

The program has been executed successfully and output is obtained.

Date:12/09/25

EXPERIMENT NO: 8

INTERMEDIATE CODE GENERATION

AIM: Implement intermediate code generation for simple expressions.

PROBLEM OBJECTIVE: To generate intermediate code for simple expressions in C.

PROBLEM DEFINITION: A C program that converts simple expressions into intermediate code, useful in the compiler design process.

ALGORITHM

1. Start
2. The expression is read.
3. Each string is read and total no. of strings in the file is calculated.
4. Each string is compared with an operator; if any operator is seen, then the previous string and next string are concatenated & stored in a temporary variable. The three-address code expression is printed.
5. Suppose the another operand is seen, then the next temporary value is concatenated to the next string using the operator & the expression is printed.
6. The final temporary value is replaced to the left operand value.
7. Stop

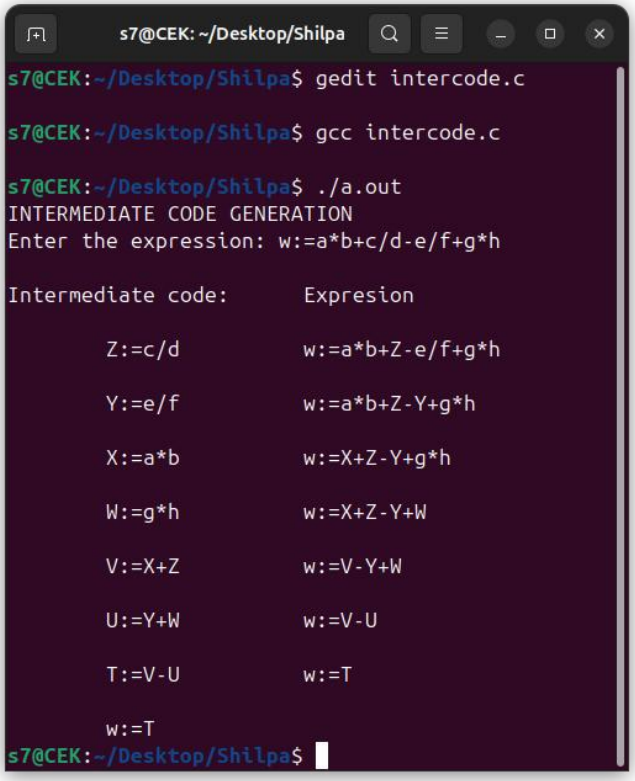
PROGRAM

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

int i = 1, j = 0, no = 0, tmpch = 90;
char str[100], left[15], right[15];

void findopr();
void explore();
void fleft(int);
void fright(int);

struct exp {
   int pos;
   char op;
} k[15];

void main() {
   printf("INTERMEDIATE CODE GENERATION\n");
   printf("Enter the expression: ");
   scanf("%s", str);
```

```
    printf("\nIntermediate code:\tExpresion\n");
    findopr();
    explore();
    printf("\n");
}

void findopr() {
    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == ':') {
            k[j].pos = i;
            k[j++].op = ':';
        }

    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '/') {
            k[j].pos = i;
            k[j++].op = '/';
        }

    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '*') {
            k[j].pos = i;
            k[j++].op = '*';
        }

    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '+') {
            k[j].pos = i;
            k[j++].op = '+';
        }

    for (i = 0; str[i] != '\0'; i++)
        if (str[i] == '-') {
            k[j].pos = i;
            k[j++].op = '-';
        }
}

void explore() {
    i = 1;
    while (k[i].op != '\0') {
        fleft(k[i].pos);
        fright(k[i].pos);

        str[k[i].pos] = tmpch--;
        printf("\t%c:=%s%c%s\t\t", str[k[i].pos], left, k[i].op, right);

        for (j = 0; j < strlen(str); j++) {
            if (str[j] != '$')
                printf("%c", str[j]);
```

```
        }
        printf("\n");
        i++;
    }

    fright(-1);
    if (no == 0) {
        fleft(strlen(str));
        printf("\t%s:=%s\n", right, left);
        exit(0);
    }

    printf("\t%s:=%c\n", right, str[k[--i].pos]);
}

void fleft(int x) {
    int w = 0, flag = 0;
    x--;

    while (x != -1 && str[x] != '+' && str[x] != '*' && str[x] != '=' &&
           str[x] != ':' && str[x] != '-' && str[x] != '/' && str[x] != '\0') {
        if (str[x] != '$' && flag == 0) {
            left[w++] = str[x];
            left[w] = '\0';
            str[x] = '$';
            flag = 1;
        }
        x--;
    }
}

void fright(int x) {
    int w = 0, flag = 0;
    x++;

    while (x != -1 && str[x] != '+' && str[x] != '*' && str[x] != '=' &&
           str[x] != ':' && str[x] != '-' && str[x] != '/' && str[x] != '\0') {
        if (str[x] != '$' && flag == 0) {
            right[w++] = str[x];
            right[w] = '\0';
            str[x] = '$';
            flag = 1;
        }
        x++;
    }
    printf("\n");
}
```

OUTPUT



RESULT

The program has been executed successfully and output is obtained.

Date:12/09/25

EXPERIMENT NO: 10

AIM: Write a lex program to display the number of lines,words and characters in an input text **.**

PROBLEM OBJECTIVE: To count lines, words, and characters in a given input using LEX.

PROBLEM DEFINITION: A LEX program that reads an input text and computes the total number of lines, words, and characters by applying lexical rules.

ALGORITHM

1. Start
2. Initialize variables:
   2.1 Initialize counter c (characters), w (words), space (spaces) and line (lines) to zero.
3. Define Lex rules:
   3.1 Whenever a space " " is encountered, increment the space counter.
   3.2 Whenever a newline "\n" is encountered, increment the line counter.
   3.3 Whenever a word [a-zA-Z0-9]+ is encountered, increment the w (word) counter and add yyleng (word length) to c (characters).
   3.4 Whenever any other character . is encountered, increment the c counter.
4. Read input:
   4.1 Use yyin = fopen("kit.txt", "r") to open the input file.
5. Process the file:
   5.1 Call yylex() to begin Lex scanning.
   5.2 Apply the defined Lex rules to scan through the file and update counters.
6. End of file condition:
   6.1 When the end of file is reached, the function yywrap() is called.
   6.2 yywrap() returns 1 to indicate end of file.
7. Print results:
   7.1 Print the total number of characters, words, spaces and lines.
8. End the program:

PROGRAM

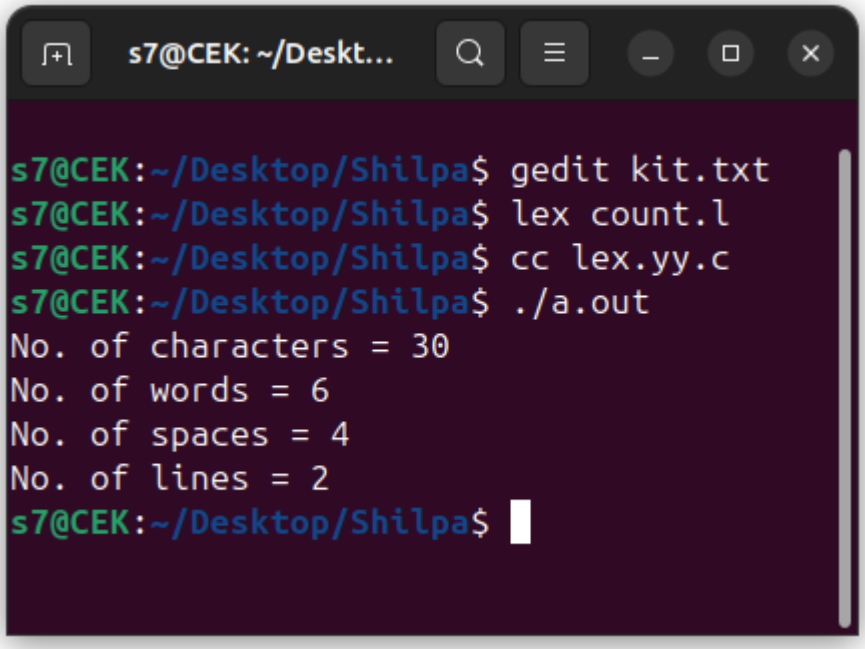
```
%{
#include <stdio.h>
int c=0, w=0, line=0, space=0;
%}

%%
[" "]     { space++; }
["\n"]    { line++; }
[a-zA-Z0-9]+  { w++; c+=yyleng; }
.         { c++; }
%%
```

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

int main()
{

   yyin = fopen("kit.txt", "r");
   yylex();
   printf("No. of characters = %d\nNo. of words = %d\nNo. of spaces = %d\nNo. of lines = %d\n", c,
w, space, line);
   return 0;
}
```

kit.txt

cat is animal
animals are creatures

OUTPUT



RESULT

The program has been executed successfully and output is obtained.

Date:12/09/25

EXPERIMENT NO: 11

B

AIM: Write a lex program to find out total number of vowels and consonants from the given input string.

PROBLEM OBJECTIVE: To count vowels and consonants in an input string using LEX.

PROBLEM DEFINITION: A LEX program that reads an input string and calculates the number of vowels and consonants based on lexical patterns.

ALGORITHM

1. Start
2. Initialize two counters → c = 0 (consonants), vowels = 0.
3. Open the input file ex.txt for reading.
4. Read input character by character using Lex rules:
   4.1 If the character is a vowel (a, e, i, o, u, A, E, I, O, U), increment vowels.
   4.2 If the character is another alphabet letter (a–z or A–Z), increment c.
   4.3 Ignore all other characters (spaces, digits, punctuation, etc.).
5. Continue step 4 until end of file is reached.
6. Stop scanning.
7. Display results:
   7.1 Print the total number of vowels and print the total number of consonants.
8. Stop.

PROGRAM

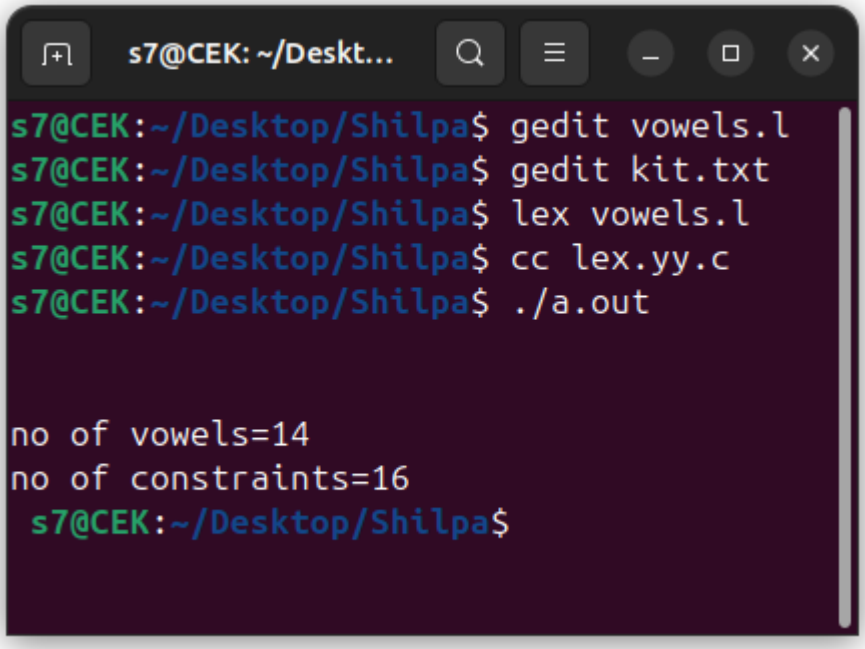
```
%{
#include <stdio.h>
int c = 0, vowels = 0;
%}

%%
[aeiouAEIOU]   { vowels++; }
[a-zA-Z]       { c++; }
%%

int yywrap() {
   return 1;
}

int main() {
   yyin = fopen("ex.txt", "r");
   yylex();
   printf("No of vowels = %d\nNo of consonants = %d\n", vowels, c - vowels);
   return 0;
}
```

ex.txt

cat is animal

animals are creatures

OUTPUT



RESULT

The program has been executed successfully and output is obtained.

STRING MANIPULATION USING LEX

AIM:
Write a LEX program to convert the substring abc to ABC from the given input string.

PROBLEM OBJECTIVE:
To implement a LEX program that detects and replaces the substring abc with ABC.

PROBLEM DEFINITION:
A LEX program that reads an input string, searches for the substring abc, and replaces it with ABC while printing the modified string as output.

ALGORITHM

1. **Start**
    - Define a global integer i to be used as an index in the loop.
2. **Lex Rules**
    - Create a pattern [a-zA-Z]* to match sequences of alphabetic characters.
3. **Iterate Over Input**
    - In the action block, iterate through the length of the matched text using a for loop.
    - Use yytext to represent the matched text and yyleng to get its length.
4. **Check for Substring "abc"**
    - For each character in the input:
        - If yytext[i] == 'a' && yytext[i+1] == 'b' && yytext[i+2] == 'c'
          → Replace them with 'A', 'B', 'C'.
5. **Output**
    - Print the modified string using printf("%s", yytext).
6. **End**
    - After processing the input string, exit the program.

PROGRAM

```
%{
#include <stdio.h>
#include <string.h>
int i;
%}

%%
[a-zA-Z]+ {
   for (i = 0; i < yyleng - 2; i++) {
      if (yytext[i] == 'a' && yytext[i+1] == 'b' && yytext[i+2] == 'c') {
         yytext[i]   = 'A';
         yytext[i+1] = 'B';
         yytext[i+2] = 'C';
      }
   }
   printf("%s", yytext);
}
```

```
\n {
   printf("\n");
   return 0;   // Exit after newline (end of one line)
}

. {
   printf("%s", yytext);   // Any other character
}
%%

int yywrap() {
   return 1;
}

int main() {
   yylex();    // Start lexer
   return 0;
}
```