Write a program in C or C++ language to find out no. of keywords, no. of operators, no. of identifiers, newlines.

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

bool isDelimiter(char ch) {
    if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
        ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
        ch == '[' || ch == ']' || ch == '{' || ch == '}')
        return true;
    return false;
}

bool isOperator(char ch) {
    if (ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=')
        return true;
    return false;
}

bool isKeyword(char *str) {
    if (!strcmp(str, "if") || !strcmp(str, "else") ||
        !strcmp(str, "while") || !strcmp(str, "do") ||
        !strcmp(str, "break") ||
        !strcmp(str, "continue") || !strcmp(str, "int")
        || !strcmp(str, "double") || !strcmp(str, "float")
        || !strcmp(str, "return") || !strcmp(str, "char")
        || !strcmp(str, "case") || !strcmp(str, "char")
        || !strcmp(str, "sizeof") || !strcmp(str, "long")
        || !strcmp(str, "short") || !strcmp(str, "typedef")
        || !strcmp(str, "switch") || !strcmp(str, "unsigned")
        || !strcmp(str, "void") || !strcmp(str, "static")
        || !strcmp(str, "struct") || !strcmp(str, "goto"))
        return true;
    return false;
}

bool validIdentifier(char *str) {
    if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
        str[0] == '3' || str[0] == '4' || str[0] == '5' ||
        str[0] == '6' || str[0] == '7' || str[0] == '8' ||
        str[0] == '9' || isDelimiter(str[0]) == true)
        return false;
    return true;
}

void parse(char *str, int *keywordCount, int *operatorCount, int *identifierCount, int *newlineCount) {
```

```
    int len = strlen(str);
    int i = 0;

    while (i < len) {
      // Skip whitespace
      while (i < len && str[i] == ' ') {
        i++;
      }

      if (isDelimiter(str[i]) == true) {
        // Check if it's an operator
        if (isOperator(str[i])) {
          (*operatorCount)++;
        }
        i++;
      } else {
        // It's not a delimiter, it's part of an identifier or keyword
        int start = i;
        while (i < len && !isDelimiter(str[i])) {
          i++;
        }
        int end = i - 1;
        char token[50]; // Assuming max token length is 50
        strncpy(token, str + start, end - start + 1);
        token[end - start + 1] = '\0';

        if (isKeyword(token)) {
          (*keywordCount)++;
        } else if (validIdentifier(token)) {
          (*identifierCount)++;
        }
      }

      // Check for newline
      if (i < len && str[i] == '\n') {
        (*newlineCount)++;
        i++;
      }
    }
  }

int main() {
  char str[100] = "int a = b + mayank; while(i<6){break;  }\n";

  int keywordCount = 0, operatorCount = 0, identifierCount = 0, newlineCount = 0;
  parse(str, &keywordCount, &operatorCount, &identifierCount, &newlineCount);

  printf("Number of keywords: %d\n", keywordCount);
  printf("Number of operators: %d\n", operatorCount);
  printf("Number of identifiers: %d\n", identifierCount);
  printf("Number of newlines: %d\n", newlineCount);

  return 0;
```

```
}
```

Write a Lex program to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.

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

%option noyywrap

%%
[ \t]+        { printf("Whitespace\n"); }
\n            { printf("Newline\n"); }
"if"|"else"|"while"|"for"|"int"|"float"|"char"|"return" { printf("Keyword: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { printf("Identifier: %s\n", yytext); }
.             { printf("Character: %s\n", yytext); }
%%

int main() {
   char input[] = "if (x > 5) {\n\tprintf(\"Hello World!\");\n}\nelse {\n\tprintf(\"Goodbye!\");\n}";
   yy_scan_string(input);
   yylex();
   return 0;
}
```

Write a program in C or C++ language to convert RE to its equivalent NFA.

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

int main() {
   char reg[20];
   int q[20][3], i = 0, j = 0, len, a, b;

   for (a = 0; a < 20; a++)
      for (b = 0; b < 3; b++)
         q[a][b] = 0;

   printf("Enter regular expression: ");
   scanf("%s", reg);
   printf("Given regular expression: %s\n", reg);
   len = strlen(reg);

   while (i < len) {
      if (reg[i] == 'a' && reg[i + 1] != '|' && reg[i + 1] != '*' && reg[i + 1] != ')') {
         q[j][0] = j + 1;
         j++;
      }
      if (reg[i] == 'b' && reg[i + 1] != '|' && reg[i + 1] != '*' && reg[i + 1] != ')') {
         q[j][1] = j + 1;
         j++;
      }
      if (reg[i] == 'e' && reg[i + 1] != '|' && reg[i + 1] != '*' && reg[i + 1] != ')') {
         q[j][2] = j + 1;
         j++;
      }
      if (reg[i] == 'a' && reg[i + 1] == '|' && reg[i + 2] == 'b') {
         q[j][2] = ((j + 1) * 10) + (j + 3);
         j++;
         q[j][0] = j + 1;
         j++;
         q[j][2] = j + 3;
         j++;
         q[j][1] = j + 1;
         j++;
         q[j][2] = j + 1;
         j++;
         i = i + 2;
      }
      if (reg[i] == 'b' && reg[i + 1] == '|' && reg[i + 2] == 'a') {
         q[j][2] = ((j + 1) * 10) + (j + 3);
         j++;
         q[j][1] = j + 1;
         j++;
         q[j][2] = j + 3;
```

```
            j++;
            q[j][0] = j + 1;
            j++;
            q[j][2] = j + 1;
            j++;
            i = i + 2;
          }
          if (reg[i] == 'a' && reg[i + 1] == '*') {
            q[j][2] = ((j + 1) * 10) + (j + 3);
            j++;
            q[j][0] = j + 1;
            j++;
            q[j][2] = ((j + 1) * 10) + (j - 1);
            j++;
          }
          if (reg[i] == 'b' && reg[i + 1] == '*') {
            q[j][2] = ((j + 1) * 10) + (j + 3);
            j++;
            q[j][1] = j + 1;
            j++;
            q[j][2] = ((j + 1) * 10) + (j - 1);
            j++;
          }
          if (reg[i] == ')' && reg[i + 1] == '*') {
            q[0][2] = ((j + 1) * 10) + 1;
            q[j][2] = ((j + 1) * 10) + 1;
            j++;
          }
          i++;
        }

        printf("\n\tTransition Table \n");
        printf("_____\n");
        printf("Current State |\tInput |\tNext State");
        printf("\n_____\n");

        for (i = 0; i <= j; i++) {
          if (q[i][0] != 0)
            printf("\n  q[%d]\t    |  a  |  q[%d]", i, q[i][0]);
          if (q[i][1] != 0)
            printf("\n  q[%d]\t    |  b  |  q[%d]", i, q[i][1]);
          if (q[i][2] != 0) {
            if (q[i][2] < 10)
              printf("\n  q[%d]\t    |  e  |  q[%d]", i, q[i][2]);
            else
              printf("\n  q[%d]\t    |  e  |  q[%d] , q[%d]", i, q[i][2] / 10, q[i][2] % 10);
          }
        }

        printf("\n_____\n");
        return 0;
}
```

Write a program in C or C++ language to convert NFA to its equivalent DFA.

```
#include <stdio.h>

int main() {
   int nfa[5][2];
   nfa[1][1] = 12;
   nfa[1][2] = 1;
   nfa[2][1] = 0;
   nfa[2][2] = 3;
   nfa[3][1] = 0;
   nfa[3][2] = 4;
   nfa[4][1] = 0;
   nfa[4][2] = 0;

   int dfa[10][2];
   int dstate[10];
   int i = 1, n, j, k, flag = 0, m, q, r;
   dstate[i++] = 1;
   n = i;

   dfa[1][1] = nfa[1][1];
   dfa[1][2] = nfa[1][2];
   printf("\nf(%d,a)=%d", dstate[1], dfa[1][1]);
   printf("\nf(%d,b)=%d", dstate[1], dfa[1][2]);

   for (j = 1; j < n; j++) {
      if (dfa[1][1] != dstate[j])
         flag++;
   }
   if (flag == n - 1) {
      dstate[i++] = dfa[1][1];
      n++;
   }
   flag = 0;
   for (j = 1; j < n; j++) {
      if (dfa[1][2] != dstate[j])
         flag++;
   }
   if (flag == n - 1) {
      dstate[i++] = dfa[1][2];
      n++;
   }
   k = 2;
   while (dstate[k] != 0) {
      m = dstate[k];
      if (m > 10) {
         q = m / 10;
         r = m % 10;
```

```
        }
        if (nfa[r][1] != 0)
            dfa[k][1] = nfa[q][1] * 10 + nfa[r][1];
        else
            dfa[k][1] = nfa[q][1];
        if (nfa[r][2] != 0)
            dfa[k][2] = nfa[q][2] * 10 + nfa[r][2];
        else
            dfa[k][2] = nfa[q][2];

        printf("\nf(%d,a)=%d", dstate[k], dfa[k][1]);
        printf("\nf(%d,b)=%d", dstate[k], dfa[k][2]);

        flag = 0;
        for (j = 1; j < n; j++) {
            if (dfa[k][1] != dstate[j])
                flag++;
        }
        if (flag == n - 1) {
            dstate[i++] = dfa[k][1];
            n++;
        }
        flag = 0;
        for (j = 1; j < n; j++) {
            if (dfa[k][2] != dstate[j])
                flag++;
        }
        if (flag == n - 1) {
            dstate[i++] = dfa[k][2];
            n++;
        }
        k++;
    }
    return 0;
}
```