Exp 1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
int id = 0, key = 0, oper = 0, del = 0, consts = 0, invalid = 0, literal = 0, header = 0;

int isKeyword(char buffer[]) {
    const char *keywords[] = {
        "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", "union",
        "unsigned", "void", "volatile", "while"
    };
    int numKeywords = sizeof(keywords) / sizeof(keywords[0]);
    for (int i = 0; i < numKeywords; ++i) {
        if (strcmp(keywords[i], buffer) == 0) {
            return 1;
        }    }
    return 0;
}

int isInvalid(char buffer[]) {
    int hasDigit = 0, hasAlpha = 0;
    for (int i = 0; i < strlen(buffer); ++i) {
        if (isdigit(buffer[i]))
            hasDigit = 1;
        if (isalpha(buffer[i]))
            hasAlpha = 1;
    }
    if (hasDigit && hasAlpha) {
        return 1;
    }
    if (buffer[0] == '#' && strlen(buffer) == 2) {
        return 1;
    }
    return 0;
}

void printToken(char buffer[]) {
    if (isKeyword(buffer)) {
        printf("(%s) keyword ", buffer);
```

```
        key++;
    } else if (isdigit(buffer[0])) {
        if (isInvalid(buffer)) {
            printf("(%s) invalid ", buffer);
            invalid++;
        } else {
            printf("(%s) const ", buffer);
            consts++;
        }
    } else if (buffer[0] == '#') {
        if (isInvalid(buffer)) {
            printf("(%s) invalid ", buffer);
            invalid++;
        } else {
            printf("(%s) header ", buffer);
            header++;
        }
    } else if (buffer[0] == '"') {
        printf("(%s) literal ", buffer);
        literal++;
    } else {
        if (isInvalid(buffer)) {
            printf("(%s) invalid ", buffer);
            invalid++;
        } else {
            printf("(%s) id ", buffer);
            id++;
        }
    }
}

void processBuffer(char buffer[], int *j) {
    if (*j != 0) {
        buffer[*j] = '\0';
        printToken(buffer);
        *j = 0;
    }
}

void printDelimiter(char ch) {
    switch (ch) {
        case ';': printf(";-delim "); break;
        case ',': printf(",-delim "); break;
        case '(': printf("(-delim "); break;
```

```
        case ')': printf(")-delim "); break;
        case '[': printf("[-delim "); break;
        case ']': printf("]-delim "); break;
        case '{': printf("{-delim "); break;
        case '}': printf("}-delim "); break;
        case '<': printf("<-delim "); break;
        case '>': printf(">-delim "); break;
    }
    del++;
}

void printOperator(char ch) {
    switch (ch) {
        case '+': printf("plus-op "); break;
        case '-': printf("minus-op "); break;
        case '*': printf("mul-op "); break;
        case '/': printf("div-op "); break;
        case '%': printf("mod-op "); break;
        case '=': printf("eq-op "); break;
    }
    oper++;}

void main() {
    char ch, buffer[100], operators[] = "+-*/%=";
    FILE *fp1;
    int i, j = 0;

    fp1 = fopen("input.txt", "r");
    if (fp1 == NULL) {
        printf("Error while opening the file\n");
        exit(0);
    }

    while ((ch = fgetc(fp1)) != EOF) {
        int isOperator = 0;

        for (i = 0; i < sizeof(operators) - 1; ++i) {
            if (ch == operators[i]) {
                processBuffer(buffer, &j);
                printOperator(ch);
                isOperator = 1;
                break;
            } }
```

```
        if (!isOperator) {
            if (isalnum(ch) || ch == '#' || ch == '.' || ch == '"' || ch == '<' || ch == '>') {
                buffer[j++] = ch;
            } else {
                processBuffer(buffer, &j);
                if (ch == ' ' || ch == '\n') {
                    if (ch == '\n') printf("\n");
                    continue;
                }
                printDelimiter(ch);}   }

        if (ch == '\n')            printf("\n");
    }

    processBuffer(buffer, &j);
    printf("\n\nKeywords: %d\nIdentifiers: %d\nOperators: %d\nDelimiters: %d\nConstants:
%d\nInvalid: %d\nLiterals: %d\nHeaders: %d\n",
            key, id, oper, del, consts, invalid, literal, header);

    fclose(fp1);
}
```

```
user@rbprojectlab05:~/jobintom$ gcc lexAnalyser.c
user@rbprojectlab05:~/jobintom$ ./a.out
(#inlcude<stdio.h>) header
(void) keyword (main) id (-delim )-delim {-delim
(int) keyword (a) id ,-delim (b) id ;-delim
(a) id eq-op (9) const ;-delim
(b) id eq-op (4) const ;-delim
(c) id eq-op (a) id div-op (b) id ;-delim
div-op div-op (this) id (is) id (a) id (test) id (code) id
div-op mul-op (Multi) id (line) id (comment) id
(second) id (line) id mul-op div-op
}-delim


Keywords: 2
Identifiers: 18
Operators: 10
Delimiters: 17
Constants: 2
Invalid: 0
Literals: 0
Headers: 1
user@rbprojectlab05:~/jobintom$
```

Open

input.txt
~/jobintom

```
#inlcude<stdio.h>
void main(){
        int a,b;
        a=9;
        b=4;
        c=a/b;
        //this is a test code
        /* Multi line comment
        second line*/
       }
```

Exp 2

```
letter [a-zA-Z]
digit[0-9]
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\tis real number",yytext);
{digit}+"."{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\t is floating pt no ",yytext);
"void"|"if"|"else"|"int"|"char"|"switch"|"return"|"struct"|"do"|"while"|"void"|"for"|"float" printf("\n%s\t
is keywords",yytext);
"\a"|"\\n"|"\\b"|"\t"|"\\t"|"\b"|"\\a" printf("\n%s\tis Escape sequences",yytext);
{letter}({letter}|{digit})* printf("\n%s\tis identifier",yytext);
"&&"|"<"|">"|"<="|">="|"="|"+"|"-"|"?"|"*"|"/"|"%"|"&"|"||" printf("\n%s\toperator ",yytext);
"{"|"}"|"["|"]"|"("|")"|"#"|""|"."|"\""|"\\"|";"|"," printf("\n%s\t is a special character",yytext);
"%d"|"%s"|"%c"|"%f"|"%e" printf("\n%s\tis a format specifier",yytext);
\n
%%
int yywrap()
{
return 1;
}
int main(void)
{
yyin=fopen("input2.txt","r");
yylex();
fclose(yyin);
return 0;
```

```
user@rbprojectlab05:~/jobintom$ lex lexInlex.l
user@rbprojectlab05:~/jobintom$ gcc lex.yy.c
user@rbprojectlab05:~/jobintom$ ./a.out

#include<stdio.h> is a preprocessor directive
int      is keywords
main    is identifier
(        is a special character
)        is a special character
{        is a special character
                is Escape sequences
int      is keywords
x       is identifier
=       operator
10      is real number
+       operator
20      is real number
;        is a special character
                is Escape sequences
printf  is identifier
(        is a special character
"        is a special character
Testing is identifier
"        is a special character
)        is a special character
;        is a special character
                is Escape sequences
\n      is Escape sequences
                is Escape sequences
return   is keywords
x       is identifier
;        is a special character
}
```

Open

input2.txt
~/jobintom

```
#include<stdio.h>

int main(){
        int x= 10 + 20;
        printf("Testing");
        \n
        return x;}
```

Exp 3

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

int lines = 0;
int words = 0;
int characters = 0;
%}

%%
\n          { lines++; characters++; }
[ \t]+      { characters += yyleng; }
[^ \t\n]+   { words++; characters += yyleng; }

%%
int main() {
    yylex();

    printf("Lines: %d\n", lines);
    printf("Words: %d\n", words);
    printf("Characters: %d\n", characters);

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

```
user@rbprojectlab05:~/jobintom$ lex count.l
user@rbprojectlab05:~/jobintom$ gcc lex.yy.c
user@rbprojectlab05:~/jobintom$ ./a.out
Hello world
This is a lex program
Lines: 2
Words: 7
Characters: 34
user@rbprojectlab05:~/jobintom$
```

Exp 4

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

%%
abc     { printf("ABC"); }
.|\n    { printf("%s", yytext); }
%%

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

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

```
user@rbprojectlab05:~/jobintom$ lex abcToABC.l
user@rbprojectlab05:~/jobintom$ gcc lex.yy.c
user@rbprojectlab05:~/jobintom$ ./a.out
This is abc news. abc news is a world famous company.
This is ABC news. ABC news is a world famous company.
user@rbprojectlab05:~/jobintom$
```

Exp 5

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

int vowels = 0;
int consonants = 0;
%}

%%
[aeiouAEIOU]    { vowels++; }
[a-zA-Z]        { consonants++; }
.|\n            { /* Ignore any other characters (digits, spaces, punctuation, etc.) */ }
%%

int main() {
    yylex();  /* Start the lexical analysis */
    printf("\nTotal number of vowels: %d\n", vowels);
    printf("Total number of consonants: %d\n", consonants);
    return 0;
}

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

```
user@rbprojectlab05:~/jobintom$ lex vowelsConsonants.l
user@rbprojectlab05:~/jobintom$ gcc lex.yy.c
user@rbprojectlab05:~/jobintom$ ./a.out
hello world
this is a sample text

Total number of vowels: 9
Total number of consonants: 18
user@rbprojectlab05:~/jobintom$
```

Exp 6

```
user@rbprojectlab05:~/jobintom$ gcc eclosure.c
user@rbprojectlab05:~/jobintom$ ./a.out
Enter the no of states: 3
Enter the states
q0

q1
q2
Epsilon closure of q0 = {  q0 q1 q2 }
Epsilon closure of q1 = {  q1 q2 }
Epsilon closure of q2 = {  q2 }
user@rbprojectlab05:~/jobintom$
```

Open

input1.txt
~/jobintom

```
q0 0 q0
q0 1 q1
q0 e q1
q1 1 q2
q1 e q2
```

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

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

void add_state(char a[3],int i){
strcpy(result[i],a); }
void display(int n){
int k=0;
printf("Epsilon 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,n,k=0;
    char state1[3],input[3],state2[3];
    printf("Enter the no of states: ");
    scanf("%d",&n);
    printf("Enter the states \n");
    for(k=0;k<n;k++){
```

```
scanf("%s",states[k]);}

for( k=0;k<n;k++){
i=0;
strcpy(state,states[k]);
strcpy(copy,state);
add_state(state,i++);
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,i++);
strcpy(state, state2);
}}}
display(i);
rewind(INPUT);
}    return 0;}
```

Exp 7

```
user@rbprojectlab05:~/pgm/jobintom$ ./a.out
Enter the number of states: 4
Enter the number of alphabets (including epsilon as 'e'): 3
Enter the alphabets (e must be last): a b e
Enter the start state: 0
Enter the number of final states: 1
Enter the final states:
3
Enter the number of transitions: 5
Enter transitions in the format: from_state symbol to_state
0 a 1
1 b 1
1 e 2
2 a 2
2 b 3
Transitions for the equivalent NFA:
From q0 on 'a': q2 q1
From q1 on 'a': q2
From q1 on 'b': q3 q2 q1
From q2 on 'a': q2
From q2 on 'b': q3
user@rbprojectlab05:~/pgm/jobintom$
```

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

```
#define MAX_STATES 20
#define MAX_ALPHABETS 20

struct Node {
	int state;
	struct Node* next;
};

int numStates, numAlphabets, startState, numFinalStates;
int finalStates[MAX_STATES];
char alphabet[MAX_ALPHABETS];
int epsilonClosure[MAX_STATES][MAX_STATES];
int closureCount[MAX_STATES];
struct Node* transitions[MAX_STATES][MAX_ALPHABETS];
struct Node* newTransitions[MAX_STATES][MAX_ALPHABETS];

void insertTransition(struct Node* transitions[MAX_STATES][MAX_ALPHABETS], int from, char
symbol, int to) {
	int index = (symbol == 'e') ? (numAlphabets - 1) : (symbol - 'a');
	struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
	newNode->state = to;
	newNode->next = transitions[from][index];
	transitions[from][index] = newNode;
}

void computeEpsilonClosure(int state, int closureIndex) {
	if (closureCount[state]) return;

	epsilonClosure[closureIndex][closureCount[closureIndex]++] = state;
	struct Node* temp = transitions[state][numAlphabets - 1]; // ε-transitions
	while (temp != NULL) {
	computeEpsilonClosure(temp->state, closureIndex);
	temp = temp->next;
	}
}

void createNFA() {
	for (int i = 0; i < numStates; i++) {
	closureCount[i] = 0; // Reset closure count
	computeEpsilonClosure(i, i);
	}

	// Now create transitions for the equivalent NFA
```

```
        for (int i = 0; i < numStates; i++) {
        for (int j = 0; j < closureCount[i]; j++) {
        int currentState = epsilonClosure[i][j];

        for (int k = 0; k < numAlphabets - 1; k++) { // Exclude 'e'
                struct Node* temp = transitions[currentState][k];
                while (temp != NULL) {
                for (int m = 0; m < closureCount[temp->state]; m++) {
                insertTransition(newTransitions, i, alphabet[k], epsilonClosure[temp->state][m]);
                }
                temp = temp->next;
                }
        }
        }
        }
}

void printNFA() {
        printf("Transitions for the equivalent NFA:\n");
        for (int i = 0; i < numStates; i++) {
        for (int j = 0; j < numAlphabets - 1; j++) { // Exclude 'e'
        struct Node* temp = newTransitions[i][j];
        if (temp != NULL) {
                printf("From q%d on '%c': ", i, alphabet[j]);
                while (temp != NULL) {
                printf("q%d ", temp->state);
                temp = temp->next;
                }
                printf("\n");
        }
        }
        }
}

int main() {
        printf("Enter the number of states: ");
        scanf("%d", &numStates);

        printf("Enter the number of alphabets (including epsilon as 'e'): ");
        scanf("%d", &numAlphabets);

        printf("Enter the alphabets (e must be last): ");
        for (int i = 0; i < numAlphabets; i++) {
        scanf(" %c", &alphabet[i]);
```

```
}
printf("Enter the start state: ");
scanf("%d", &startState);
printf("Enter the number of final states: ");
scanf("%d", &numFinalStates);

printf("Enter the final states:\n");
for (int i = 0; i < numFinalStates; i++) {
scanf("%d", &finalStates[i]);
}

int numTransitions;
printf("Enter the number of transitions: ");
scanf("%d", &numTransitions);

printf("Enter transitions in the format: from_state symbol to_state\n");
for (int i = 0; i < numTransitions; i++) {
int fromState, toState;
char symbol;
scanf("%d %c %d", &fromState, &symbol, &toState);
insertTransition(transitions, fromState, symbol, toState);
}
for (int i = 0; i < numStates; i++) {
closureCount[i] = 0;
}

createNFA();


printNFA();

for (int i = 0; i < numStates; i++) {
for (int j = 0; j < numAlphabets; j++) {
struct Node* temp = transitions[i][j];
while (temp) {
        struct Node* toDelete = temp;
        temp = temp->next;
        free(toDelete);
}}}
return 0;}
```