Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LEN 100

const char \*keywords[] = {"if", "else", "while", "return",

"int", "float", "void"};

const int keywordCount = sizeof(keywords) / sizeof(keywords[0]);

int isKeyword(char \*lexeme) {

if (lexeme == NULL) return 0;

for (int i = 0; i < keywordCount; i++) if (strcmp(lexeme, keywords[i]) == 0) return 1;

return 0;

}

int isDelimiter(char ch) {

return (ch == ';' || ch == ',' || ch == '(' || ch == ')' || ch == '{' ||

ch == '}' || ch == '[' || ch == ']');

}

int isOperator(char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '=' ||

ch == '<' || ch == '>' || ch == '!' || ch == '&' || ch == '|' ||

ch == '%' || ch == '^' || ch == '~' || ch == '?' || ch == ':');

}

void skipSingleLineComment(FILE \*fp) {

int ch;

while ((ch = fgetc(fp)) != EOF && ch != '\n') {}

if (ch == '\n') ungetc(ch, fp);

}

void skipMultiLineComment(FILE \*fp) {

int ch, prev = 0;

while ((ch = fgetc(fp)) != EOF) {

if (prev == '\*' && ch == '/') break;

prev = ch;

}

}

void lexer(FILE \*fp, FILE \*op) {

int c, i,tokenCount = 0;

char buffer[MAX\_LEN];

while ((c = fgetc(fp)) != EOF) {

if (isspace(c)) continue;

if (isalpha(c) || c == '\_') {

i = 0;

buffer[i++] = c;

while (i < MAX\_LEN - 1 && (isalnum(c = fgetc(fp)) || c == '\_')) {

buffer[i++] = c;

}

ungetc(c, fp);

buffer[i] = '\0';

if (isKeyword(buffer)) {

fprintf(op, "<KEYWORD, %s>\n", buffer);

} else {

fprintf(op, "<IDENTIFIER, %s>\n", buffer);

}

tokenCount++;

}

else if (isdigit(c)) {

i = 0;

int hasDot = 0;

buffer[i++] = c;

while (i < MAX\_LEN - 1 && (c = fgetc(fp)) != EOF) {

if (isdigit(c)) {

buffer[i++] = c;

} else if (c == '.' && !hasDot) {

hasDot = 1;

buffer[i++] = c;

} else {

ungetc(c, fp);

break;

}

}

buffer[i] = '\0';

fprintf(op, "<NUMBER, %s>\n", buffer);

tokenCount++;

}

else if (c == '"') {

i = 0;

buffer[i++] = c;

while (i < MAX\_LEN - 1 && (c = fgetc(fp)) != EOF) {

buffer[i++] = c;

if (c == '"')

break;

if (c == '\\') {

if (i < MAX\_LEN - 1 && (c = fgetc(fp)) != EOF) {

buffer[i++] = c;

}

}

}

buffer[i] = '\0';

fprintf(op, "<STRING, %s>\n", buffer);

tokenCount++;

}

else if (c == '\'') {

i = 0;

buffer[i++] = c;

while (i < MAX\_LEN - 1 && (c = fgetc(fp)) != EOF) {

buffer[i++] = c;

if (c == '\'') break;

if (c == '\\') {

if (i < MAX\_LEN - 1 && (c = fgetc(fp)) != EOF) {

buffer[i++] = c;

}

}

}

buffer[i] = '\0';

fprintf(op, "<CHARACTER, %s>\n", buffer);

tokenCount++;

}

else if (isOperator(c)) {

int next = fgetc(fp);

if (next != EOF) {

if ((c == '=' && next == '=') || (c == '!' && next == '=') ||

(c == '<' && next == '=') || (c == '>' && next == '=') ||

(c == '&' && next == '&') || (c == '|' && next == '|') ||

(c == '+' && next == '+') || (c == '-' && next == '-') ||

(c == '+' && next == '=') || (c == '-' && next == '=') ||

(c == '\*' && next == '=') || (c == '/' && next == '=') ||

(c == '%' && next == '=') || (c == '<' && next == '<') ||

(c == '>' && next == '>') || (c == '-' && next == '>')) {

fprintf(op, "<OPERATOR, %c%c>\n", c, next);

} else {

ungetc(next, fp);

fprintf(op, "<OPERATOR, %c>\n", c);

}

} else {

fprintf(op, "<OPERATOR, %c>\n", c);

}

tokenCount++;

}

else if (isDelimiter(c)) {

fprintf(op, "<DELIMITER, %c>\n", c);

tokenCount++;

}

else if (c == '#') {

i = 0;

buffer[i++] = c;

while (i < MAX\_LEN - 1 && (c = fgetc(fp)) != EOF) {

if (c == '\n') {

ungetc(c, fp);

break;

}

buffer[i++] = c;

}

buffer[i] = '\0';

fprintf(op, "<PREPROCESSOR, %s>\n", buffer);

tokenCount++;

}

}

printf("Total tokens: %d\n", tokenCount);

}

int main() {

FILE \*fp = fopen("input.txt", "r");

if (fp == NULL) {

printf("Error opening file\n");

return 1;

}

FILE \*op = fopen("output.txt", "w");

lexer(fp, op);

fclose(fp);

fclose(op);

return 0;

}

**INPUT:**

if main(){

int res = 0;

if (res == 0){

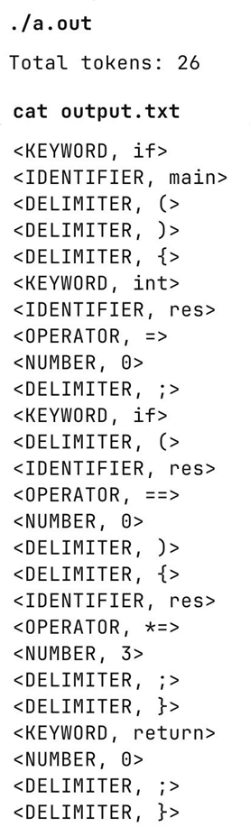
res \*= 3;

}

return 0;

}

**OUTPUT:**



Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

%{

#include <stdio.h>

%}

%%

.\*vina.\* {printf("Contains first four characters as substring \n");}

.\* {printf("Accepted \n");}

%%

int yywrap(){

return 1;

}

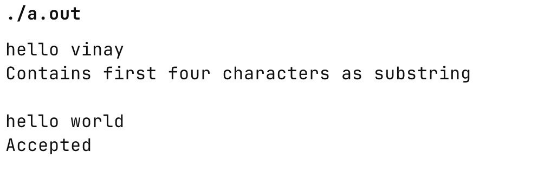
int main(){

yylex();

return 0;

}

**OUTPUT:**



Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

**program.l**

%{

#include "program.tab.h"

%}

%%

[a-zA-Z\_][a-zA-Z\_0-9]\* return letter;

[0-9] return digit;

. return yytext[0];

\n return 0;

%%

int yywrap() {

return 1;

}

**program.y**

%{

#include <stdio.h>

extern int yylex();

extern int yyerror(const char \*);

%}

%token digit letter

%%

start : letter s {printf("Valid variable\n");}

;

s : letter s

| digit s

| /\* empty \*/

;

%%

int yyerror(const char \*s) {

printf("\nIt's not an identifier!\n");

return 0;

}

int main() {

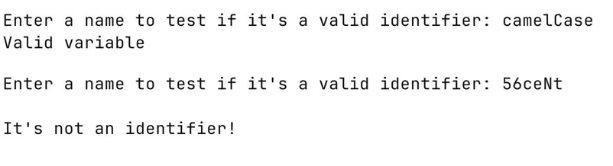
printf("\nEnter a name to test if it's a valid identifier: ");

yyparse();

return 0;

}

**OUTPUT:**

****

Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

**calculator.l**

%{

#include "calculator.tab.h"

#include <stdio.h>

%}

%option noyywrap

%%

[0-9]+ {yylval = atoi(yytext); return NUMBER;}

[\t\r]|" " ;

\n {return '\n';}

"+" {return PLUS;}

"-" {return MINUS;}

"\*" {return MUL;}

"/" {return DIV;}

"(" {return LPAREN;}

")" {return RPAREN;}

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

%%

**calculator.y**

%{

#include <stdio.h>

#include <stdlib.h>

int yylex(void);

void yyerror(const char \*s);

%}

%token NUMBER PLUS MINUS MUL DIV LPAREN RPAREN

%left PLUS MINUS

%left MUL DIV

%right UMINUS

%%

input:

//empty

| input expr '\n' { printf("= %d\nEnter the expression: ",$2);}

;

expr:

NUMBER

| expr PLUS expr {$$ = $1 + $3;}

| expr MINUS expr {$$ = $1 - $3;}

| expr MUL expr {$$ = $1 \* $3;}

| expr DIV expr {

if ($3 == 0) {

yyerror("Division by zero");

$$ = 0;

} else {

$$ = $1 / $3;

}

}

| LPAREN expr RPAREN {$$ = $2;}

| MINUS expr %prec UMINUS { $$ = -$2; }

;

%%

int main(){

printf("Enter the expression: ");

yyparse();

return 0;

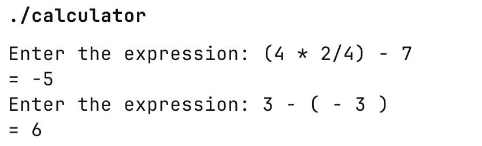
}

void yyerror(const char \*s){

fprintf(stderr, "Error: %s\n",s);

}

**OUTPUT:**



Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

**bnf.l**

%{

#include "bnf.tab.h"

%}

%%

[0-9]+ { yylval.num = atoi(yytext); return NUMBER; }

[ \t]+ ; /\* ignore spaces and tabs \*/

\n ; /\* ignore newlines \*/

. { return yytext[0]; } /\* return single-character operators like + - \* / ( ) \*/

%%

**bnf.y**

%code requires {

typedef struct AST {

char \*nodeType;

int value;

struct AST \*left, \*right;

} AST;

}

%union {

int num;

AST \*node;

}

%token <num> NUMBER

%type <node> expr

%left '+' '-'

%left '\*' '/'

%left UMINUS

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int yylex(void);

int yyerror(const char \*s);

AST\* makeNode(char \*type, AST \*l, AST \*r);

AST\* makeNum(int val);

void printAST(AST \*root, int lvl);

int evalAST(AST \*root);

%}

%%

start: expr {

printf("\nAST:\n");

printAST($1,0);

printf("\nResult: %d\n", evalAST($1));

return 0;

};

expr: expr '+' expr { $$ = makeNode("+",$1,$3); }

| expr '-' expr { $$ = makeNode("-",$1,$3); }

| expr '\*' expr { $$ = makeNode("\*",$1,$3); }

| expr '/' expr { $$ = makeNode("/",$1,$3); }

| '-' expr %prec UMINUS { $$ = makeNode("NEG",$2,NULL); }

| '(' expr ')' { $$ = $2; }

| NUMBER { $$ = makeNum($1); }

;

%%

AST\* makeNode(char \*type, AST \*l, AST \*r) {

AST \*n = malloc(sizeof(AST));

n->nodeType = strdup(type);

n->value = 0;

n->left = l;

n->right = r;

return n;

}

AST\* makeNum(int val) {

AST \*n = malloc(sizeof(AST));

n->nodeType = strdup("NUM");

n->value = val;

n->left = n->right = NULL;

return n;

}

void printAST(AST \*root, int lvl) {

if (!root) return;

for (int i = 0; i < lvl; i++) printf(" ");

if (strcmp(root->nodeType, "NUM") == 0)

printf("NUM(%d)\n", root->value);

else

printf("%s\n", root->nodeType);

printAST(root->left, lvl + 1);

printAST(root->right, lvl + 1);

}

int evalAST(AST \*root) {

if (!root) return 0;

if (strcmp(root->nodeType, "NUM") == 0) return root->value;

if (strcmp(root->nodeType, "+") == 0) return evalAST(root->left) + evalAST(root->right);

if (strcmp(root->nodeType, "-") == 0) return evalAST(root->left) - evalAST(root->right);

if (strcmp(root->nodeType, "\*") == 0) return evalAST(root->left) \* evalAST(root->right);

if (strcmp(root->nodeType, "/") == 0) return evalAST(root->left) / evalAST(root->right);

if (strcmp(root->nodeType, "NEG") == 0) return -evalAST(root->left);

return 0;

}

int main() {

printf("Enter arithmetic expression:\n");

yyparse();

return 0;

}

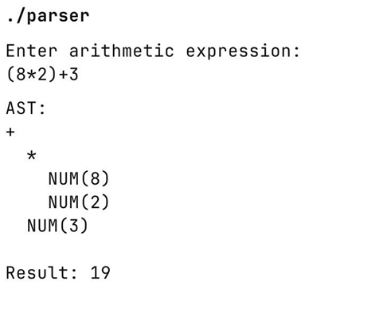
int yyerror(const char \*s) {

printf("Error: %s\n", s);

return 0;

}

**OUTPUT:**



Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

**for.l**

%{

#include <string.h>

#include "for.tab.h"

char \*datatypes[] = {"int","char","float","double"};

int size = sizeof(datatypes)/sizeof(datatypes[0]);

%}

%option noyywrap

%%

"for" { return FOR; }

"(" { return LPAREN; }

")" { return RPAREN; }

";" { return SEMICOLON;}

"=" {return ASSIGN;}

"<"|">"|"<="|">="|"!="|"==" { return COMPARE;}

"+="|"-="|"\*="|"/=" { return UPDATE;}

"--"|"++" { return UNARY\_OP;}

[0-9]+ { yylval = atoi(yytext); return NUMBER; }

[a-zA-Z\_][a-zA-Z0-9\_]\* { for(int i=0;i<size;i++) {

if(strcmp(yytext,datatypes[i])==0)

return DATATYPE;

}

return IDENTIFIER; }

[ \t\n] ; /\* ignore whitespace \*/

. { return yytext[0]; }

%%

**for.y**

%{

#include <stdio.h>

#include <string.h>

int yylex();

void yyerror(const char \*s);

%}

%token FOR RPAREN LPAREN NUMBER IDENTIFIER SEMICOLON DATATYPE UPDATE ASSIGN COMPARE UNARY\_OP

%%

start: FOR LPAREN i\_expr SEMICOLON c\_expr SEMICOLON u\_expr RPAREN { printf("Valid for loop\n");return 0; }

;

c\_expr: IDENTIFIER COMPARE IDENTIFIER

| IDENTIFIER COMPARE NUMBER

;

u\_expr: IDENTIFIER UPDATE NUMBER

| IDENTIFIER UPDATE IDENTIFIER

| IDENTIFIER UNARY\_OP

i\_expr: variable ASSIGN NUMBER

| variable ASSIGN IDENTIFIER

;

variable: DATATYPE IDENTIFIER

| IDENTIFIER

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Invalid for loop\n");

}

int main(){

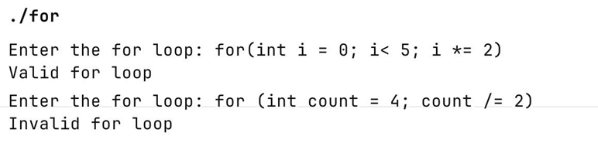
printf("Enter the for loop: ");

yyparse();

return 0;

}

**OUTPUT:**



Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct States {

char state[5];

int closure\_count;

char closures[50][5];

} recorded\_states[50];

struct Transition {

char from[5];

char to[5];

} eps\_trans[100];

int count = 0;

int eps\_count = 0;

int search\_state(char \*initial) {

for (int i = 0; i < count; i++) if (strcmp(recorded\_states[i].state, initial) == 0) return i;

return -1;

}

int add\_state(char \*initial) {

strcpy(recorded\_states[count].state, initial);

recorded\_states[count].closure\_count = 1;

strcpy(recorded\_states[count].closures[0], initial);

return count++;

}

int in\_closure(int i, char \*state) {

for (int x = 0; x < recorded\_states[i].closure\_count; x++)

if (strcmp(recorded\_states[i].closures[x], state) == 0) return 1;

return 0;

}

void add\_closure(char \*initial, char \*final) {

int i = search\_state(initial);

if (i == -1) i = add\_state(initial);

if (strcmp(final, "") != 0 && !in\_closure(i, final)) strcpy(recorded\_states[i].closures[recorded\_states[i].closure\_count++], final);

}

void expand\_closures() {

int changed = 1;

while (changed) {

changed = 0;

for (int i = 0; i < count; i++) {

for (int x = 0; x < recorded\_states[i].closure\_count; x++) {

char \*state = recorded\_states[i].closures[x];

// Check epsilon transitions from this state

for (int t = 0; t < eps\_count; t++) {

if (strcmp(eps\_trans[t].from, state) == 0) {

if (!in\_closure(i, eps\_trans[t].to)) {

strcpy(recorded\_states[i].closures[recorded\_states[i].closure\_count++],

eps\_trans[t].to);

changed = 1;

} } }

} } } }

void find\_closures(char \*filename) {

FILE \*fp = fopen(filename, "r");

if (!fp) {

printf("No such file exists\n");

exit(0);

}

char initial[5], final[5], symbol[5];

while (fscanf(fp, "%s %s %s", initial, symbol, final) == 3) {

if (strcmp(symbol, "e") == 0) {

add\_closure(initial, final);

strcpy(eps\_trans[eps\_count].from, initial);

strcpy(eps\_trans[eps\_count].to, final);

eps\_count++;

}

add\_closure(initial, "");

add\_closure(final, "");

}

fclose(fp);

expand\_closures();

}

int main() {

char filename[50];

printf("Enter the name of the file with state transitions: ");

scanf("%s", filename);

find\_closures(filename);

printf("Epsilon Closures of all states:\n");

for (int i = 0; i < count; i++) {

printf("%s: {", recorded\_states[i].state);

for (int x = 0; x < recorded\_states[i].closure\_count; x++) {

printf("%s%s", recorded\_states[i].closures[x],

x + 1 == recorded\_states[i].closure\_count ? "" : ", ");

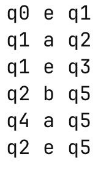
}

printf("}\n");

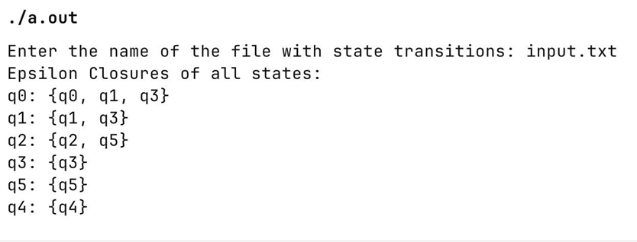
}

}

**INPUT:**

****

**OUTPUT:**



Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct States {

char state[5];

int closure\_count;

char closures[50][5];

} recorded\_states[50];

struct Transition {

char from[5];

char symbol[5];

char to[5];

} trans[100];

struct Transition eps\_trans[100]; // Only epsilon

int count = 0, t\_count = 0, eps\_count = 0;

int search\_state(char \*initial) {

for (int i = 0; i < count; i++) if (strcmp(recorded\_states[i].state, initial) == 0) return i;

return -1;

}

int add\_state(char \*initial) {

strcpy(recorded\_states[count].state, initial);

recorded\_states[count].closure\_count = 1;

strcpy(recorded\_states[count].closures[0], initial);

return count++;

}

int in\_closure(int i, char \*state) {

for (int x = 0; x < recorded\_states[i].closure\_count; x++)

if (strcmp(recorded\_states[i].closures[x], state) == 0) return 1;

return 0;

}

void add\_closure(char \*initial, char \*final) {

int i = search\_state(initial);

if (i == -1) i = add\_state(initial);

if (strcmp(final, "") != 0 && !in\_closure(i, final)) strcpy(recorded\_states[i].closures[recorded\_states[i].closure\_count++],final);

}

void expand\_closures() {

int changed = 1;

while (changed) {

changed = 0;

for (int i = 0; i < count; i++) {

for (int x = 0; x < recorded\_states[i].closure\_count; x++) {

char \*state = recorded\_states[i].closures[x];

for (int t = 0; t < eps\_count; t++) {

if (strcmp(eps\_trans[t].from, state) == 0) {

if (!in\_closure(i, eps\_trans[t].to)) {

strcpy(recorded\_states[i]

.closures[recorded\_states[i].closure\_count++],

eps\_trans[t].to);

changed = 1;

} } } }

} } }

void find\_closures(char \*filename) {

FILE \*fp = fopen(filename, "r");

if (!fp) {

printf("No such file exists\n");

exit(0);

}

char initial[5], final[5], symbol[5];

while (fscanf(fp, "%s %s %s", initial, symbol, final) == 3) {

if (strcmp(symbol, "e") == 0) { // epsilon

add\_closure(initial, final);

strcpy(eps\_trans[eps\_count].from, initial);

strcpy(eps\_trans[eps\_count].to, final);

eps\_count++;

} else {

strcpy(trans[t\_count].from, initial);

strcpy(trans[t\_count].symbol, symbol);

strcpy(trans[t\_count].to, final);

t\_count++;

}

add\_closure(initial, "");

add\_closure(final, "");

}

fclose(fp);

expand\_closures();

}

int in\_set(char result[50][5], int \*rcount, char \*s) {

for (int i = 0; i < \*rcount; i++) if (strcmp(result[i], s) == 0) return 1;

return 0;

}

void move\_closure(int state\_index, char \*symbol, char result[50][5],

int \*rcount) {

\*rcount = 0;

for (int i = 0; i < recorded\_states[state\_index].closure\_count; i++) {

char \*s = recorded\_states[state\_index].closures[i];

for (int t = 0; t < t\_count; t++) {

if (strcmp(trans[t].from, s) == 0 &&

strcmp(trans[t].symbol, symbol) == 0) {

int idx = search\_state(trans[t].to);

if (idx != -1) {

for (int c = 0; c < recorded\_states[idx].closure\_count; c++) {

if (!in\_set(result, rcount, recorded\_states[idx].closures[c])) {

strcpy(result[(\*rcount)++], recorded\_states[idx].closures[c]);

} } } }

} } }

int main() {

char filename[50];

printf("Enter the name of the file with state transitions: ");

scanf("%s", filename);

find\_closures(filename);

printf("\nEpsilon Closures of all states:\n");

for (int i = 0; i < count; i++) {

printf("%s: {", recorded\_states[i].state);

for (int x = 0; x < recorded\_states[i].closure\_count; x++) {

printf("%s%s", recorded\_states[i].closures[x],

x + 1 == recorded\_states[i].closure\_count ? "" : ", ");

}

printf("}\n");

}

char all\_symbols[20][5];

int all\_scount = 0;

for (int t = 0; t < t\_count; t++) {

int found = 0;

for (int k = 0; k < all\_scount; k++) {

if (strcmp(all\_symbols[k], trans[t].symbol) == 0) {

found = 1;

break;

}

}

if (!found) strcpy(all\_symbols[all\_scount++], trans[t].symbol);

}

printf("\nEquivalent NFA Transitions (without epsilon):\n");

for (int i = 0; i < count; i++) {

for (int k = 0; k < all\_scount; k++) {

char result[50][5];

int rcount;

move\_closure(i, all\_symbols[k], result, &rcount);

printf("%s --%s--> { ", recorded\_states[i].state, all\_symbols[k]);

for (int r = 0; r < rcount; r++) {

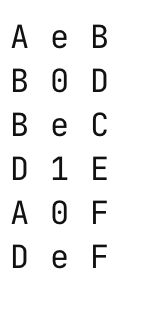
printf("%s%s", result[r], r + 1 == rcount ? "" : ", ");

}

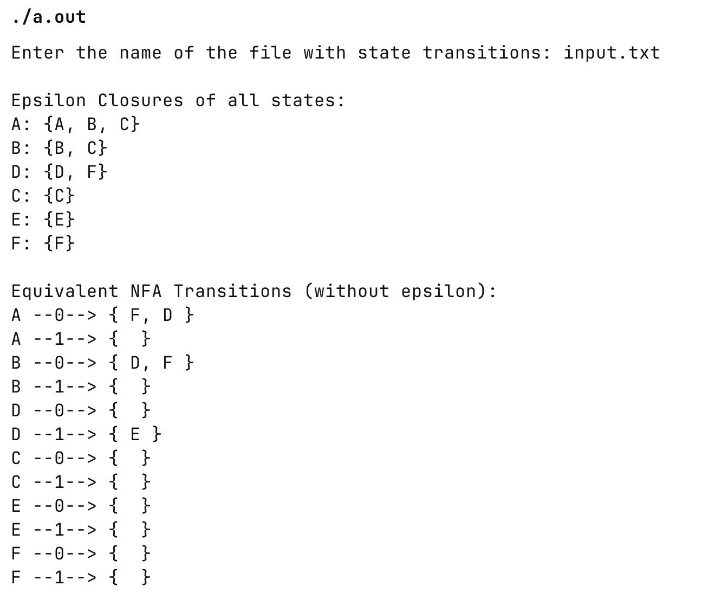
printf(" }\n");

} } }

**INPUT:**



**OUTPUT:**



Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STATES 100

char sym[5], nfastate[5], symbols[100][5];

int curr\_state\_count, res\_count, symbol\_count = 0;

char nfa\_final\_states[MAX\_STATES][5];

int nfa\_final\_count = 0;

struct NFATransition {

char from[5], symbol[5], to[5];

} nfa\_trans[100];

int nfa\_count = 0;

struct DFATransition {

char from[MAX\_STATES][5], symbol[5], to[MAX\_STATES][5];

int from\_count, to\_count;

} dfa\_trans[100];

int dfa\_count = 0;

struct DFAState {

char state[MAX\_STATES][5];

int state\_count;

} dfa\_states[100];

int dfa\_state\_count = 0;

int search\_state(char \*state) for (int i = 0; i < nfa\_count; i++) {

if (strcmp(nfa\_trans[i].from, state) == 0) return i;

return -1;

}

void add\_to\_res(char \*state, char \*symbol, char res[][5], int \*res\_count) {

for (int i = 0; i < nfa\_count; i++)

if (strcmp(nfa\_trans[i].from, state) == 0 &&

strcmp(nfa\_trans[i].symbol, symbol) == 0) strcpy(res[(\*res\_count)++], nfa\_trans[i].to);

}

int is\_dfa\_final(struct DFAState dfa\_state) {

for (int i = 0; i < dfa\_state.state\_count; i++)

for (int j = 0; j < nfa\_final\_count; j++) if (strcmp(dfa\_state.state[i], nfa\_final\_states[j]) == 0) return 1;

return 0;

}

void find\_nfa\_transitions(char \*filename) {

FILE \*fp = fopen(filename, "r");

if (!fp) {

printf("No such file exists\n");

exit(0);

}

char initial[5], final[5], symbol[5];

while (fscanf(fp, "%s %s %s", initial, symbol, final) == 3) {

strcpy(nfa\_trans[nfa\_count].from, initial);

strcpy(nfa\_trans[nfa\_count].symbol, symbol);

strcpy(nfa\_trans[nfa\_count++].to, final);

if (symbol\_count == 0)

strcpy(symbols[symbol\_count++], symbol);

else {

int found = 0;

for (int i = 0; i < symbol\_count; i++) {

if (strcmp(symbols[i], symbol) == 0) {

found = 1;

break;

}

}

if (!found)

strcpy(symbols[symbol\_count++], symbol);

}

}

fclose(fp);

}

int state\_exists(char res[][5], int res\_count) {

for (int i = 0; i < dfa\_state\_count; i++) {

if (dfa\_states[i].state\_count == res\_count) {

int match = 1;

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

int found = 0;

for (int k = 0; k < res\_count; k++) {

if (strcmp(res[j], dfa\_states[i].state[k]) == 0) {

found = 1;

break;

}

}

if (!found) {

match = 0;

break;

}

}

if (match)

return i;

}

}

return -1;

}

void add\_dfa\_transition(char from[][5], int from\_count, char \*symbol,

char to[][5], int to\_count) {

for (int i = 0; i < from\_count; i++)

strcpy(dfa\_trans[dfa\_count].from[i], from[i]);

for (int i = 0; i < to\_count; i++)

strcpy(dfa\_trans[dfa\_count].to[i], to[i]);

strcpy(dfa\_trans[dfa\_count].symbol, symbol);

dfa\_trans[dfa\_count].from\_count = from\_count;

dfa\_trans[dfa\_count++].to\_count = to\_count;

}

void print\_transitions() {

printf("\nDFA Transitions:\n");

for (int i = 0; i < dfa\_count; i++) {

printf("{ ");

for (int j = 0; j < dfa\_trans[i].from\_count; j++)

printf("%s ", dfa\_trans[i].from[j]);

printf("} --%s--> {", dfa\_trans[i].symbol);

for (int j = 0; j < dfa\_trans[i].to\_count; j++)

printf(" %s", dfa\_trans[i].to[j]);

printf(" }\n");

}

}

void find\_dfa\_transistions(char \*start\_state) {

strcpy(dfa\_states[dfa\_state\_count++].state[0], start\_state);

dfa\_states[0].state\_count = 1;

for (int curr\_state = 0; curr\_state < dfa\_state\_count; curr\_state++) {

for (int s = 0; s < symbol\_count; s++) {

char res[MAX\_STATES][5];

res\_count = 0;

strcpy(sym, symbols[s]);

curr\_state\_count = dfa\_states[curr\_state].state\_count;

for (int st = 0; st < curr\_state\_count; st++) {

strcpy(nfastate, dfa\_states[curr\_state].state[st]);

add\_to\_res(nfastate, sym, res, &res\_count);

}

if (state\_exists(res, res\_count) == -1 && res\_count > 0) {

for (int r = 0; r < res\_count; r++)

strcpy(dfa\_states[dfa\_state\_count].state[r], res[r]);

dfa\_states[dfa\_state\_count].state\_count = res\_count;

dfa\_state\_count++;

}

add\_dfa\_transition(dfa\_states[curr\_state].state,

dfa\_states[curr\_state].state\_count, sym, res,

res\_count);

}

}

print\_transitions();

}

int main() {

char filename[50];

printf("Enter the name of the file with state transitions: ");

scanf("%s", filename);

find\_nfa\_transitions(filename);

printf("Enter the start state of the NFA: ");

char start\_state[5];

char fin\_state[5];

scanf("%s", start\_state);

// check if start state is valid

if (search\_state(start\_state) == -1) {

printf("Invalid start state\n");

exit(0);

}

printf("Enter number of final states in NFA: ");

scanf("%d", &nfa\_final\_count);

printf("Enter the final states: ");

for (int i = 0; i < nfa\_final\_count; i++) {

scanf("%s", fin\_state);

if (search\_state(fin\_state) == -1) {

printf("Invalid final state: %s\n", fin\_state);

exit(0);

} else strcpy(nfa\_final\_states[i], fin\_state);

}

find\_dfa\_transistions(start\_state);

printf("\nDFA Final States:\n");

for (int i = 0; i < dfa\_state\_count; i++) {

if (is\_dfa\_final(dfa\_states[i])) {

printf("{ ");

for (int j = 0; j < dfa\_states[i].state\_count; j++) {

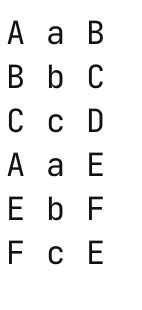
printf("%s ", dfa\_states[i].state[j]);

}

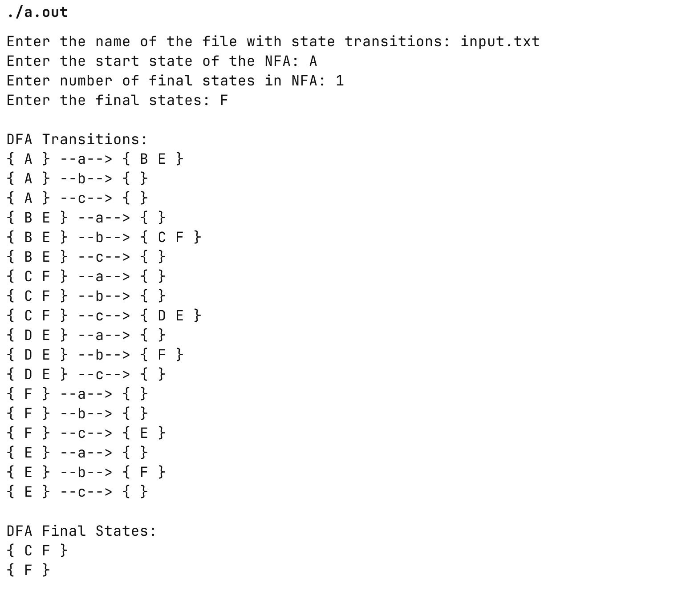
printf("}\n");

} } }

**INPUT:**



**OUTPUT:**

****

Name: Vinay Chandrasekhar Date:

Roll no: 63

**PROGRAM:**

#include <stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 50

int transition[MAX][MAX];

int states, symbols;

int final\_states[MAX], final\_count;

int start\_state;

int partition[MAX], new\_partition[MAX];

int main() {

printf("Enter number of states: ");

scanf("%d", &states);

printf("Enter number of symbols: ");

scanf("%d", &symbols);

printf("Enter transition table (%d x %d):\n", states, symbols);

for (int i = 0; i < states; i++)

for (int j = 0; j < symbols; j++) scanf("%d", &transition[i][j]);

printf("Enter start state: ");

scanf("%d", &start\_state);

printf("Enter number of final states: ");

scanf("%d", &final\_count);

memset(final\_states, 0, sizeof(final\_states));

printf("Enter final states: ");

for (int i = 0; i < final\_count; i++) {

int f;

scanf("%d", &f);

final\_states[f] = 1;

}

for (int i = 0; i < states; i++) partition[i] = final\_states[i] ? 1 : 0;

int changed;

do {

changed = 0;

int new\_class = 0;

for (int i = 0; i < states; i++)

new\_partition[i] = -1;

for (int i = 0; i < states; i++) {

if (new\_partition[i] == -1) {

new\_partition[i] = new\_class;

for (int j = i + 1; j < states; j++) {

if (partition[i] == partition[j]) {

int same = 1;

for (int s = 0; s < symbols; s++) {

if (partition[transition[i][s]] != partition[transition[j][s]]) {

same = 0;

break;

}

}

if (same) new\_partition[j] = new\_class;

}

}

new\_class++;

}

}

for (int i = 0; i < states; i++) {

if (partition[i] != new\_partition[i]) {

changed = 1;

break;

} }

for (int i = 0; i < states; i++) partition[i] = new\_partition[i];

} while (changed);

int classes = 0;

for (int i = 0; i < states; i++) {

if (partition[i] > classes)

classes = partition[i];

}

classes++;

printf("\nMinimized DFA has %d states:\n", classes);

for (int i = 0; i < classes; i++) {

printf("State %d: {", i);

for (int j = 0; j < states; j++) if (partition[j] == i) printf(" %d ", j);

printf("}\n");

}

printf("\nTransitions in Minimized DFA:\n");

for (int i = 0; i < classes; i++) {

for (int s = 0; s < symbols; s++) {

int old\_rep = -1;

for (int j = 0; j < states; j++) {

if (partition[j] == i) {

old\_rep = j;

break;

} }

if (old\_rep != -1) printf("%d --%d--> %d\n", i, s, partition[transition[old\_rep][s]]);

} }

printf("\nStart state: %d\n", partition[start\_state]);

printf("Final states: ");

for (int i = 0; i < states; i++) if (final\_states[i] && partition[i] != -1)

printf("%d ", partition[i]);

printf("\n");

return 0;

}

**OUTPUT:**

