Program Code

Lexical.c

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

#include "functions.h"

#define file "input.c"

#define debug(x) puts(x)
static int comment_flag = 0;
// #define debug(x) ;

/*Hello World
Good Morning*/

FILE *fp;

int handle_line(char *line) {
  char line_copy[1024];
  strncpy(line_copy, line, 1024);
  int lit_flag = 0;
  int lit_type = 1;  // 1- "  &&  2-'
  char lexeme[1024] = "";
  for (int i = 0; i < strlen(line); i++) {
    char curr_char = line[i];

    // handling multiline comment
    if (i + 1 < strlen(line) && (line[i] == '/' && line[i + 1] ==
'*')) {
      comment_flag = 1;
      fprintf(fp,"multiline comment");
    }
    if (i + 1 < strlen(line) && (line[i] == '*' && line[i + 1] ==
'/')) {
      comment_flag = 0;
      fprintf(fp,"multiline comment");
      lexeme[0]='\0';
      break;
    }
    // if still inside comment
    if (comment_flag == 1) {
      continue;
    }

    // handling single line comment
    if (i + 1 < strlen(line) && (line[i] == '/' && line[i + 1] ==
'/')) {
      fprintf(fp,"Single line comment");
      break;
    }

    // handling string literals
    if (strlen(lexeme) < 1 && lit_flag == 0) {
      if (curr_char == '"') {
        lit_type = 1;
        lit_flag = 1;

      } else if (curr_char == '\'') {
        lit_type = 2;
        lit_flag = 1;
      }
    } else if (lit_flag == 1 && ((lit_type == 1 && curr_char ==
'"') || (lit_type == 2 && curr_char == '\''))) {
      lit_flag = 0;
      int len = strlen(lexeme);
      lexeme[len] = curr_char;
      lexeme[len + 1] = '\0';

      // changes made to print literals and " seperately
      if (lexeme[len] == '"') {
        fprintf(fp,"<%s,%s>\n", "\"", "symbol");
        for (int i = 0; i < len + 1; i++) {
          lexeme[i] = lexeme[i + 1];
        }
        lexeme[len - 1] = '\0';
        fprintf(fp,"<%s,%s>\n", lexeme, "literal");
        fprintf(fp,"<%s,%s>\n", "\"", "symbol");
      }

      // fprintf(fp,"<%s,%s>\n", lexeme, "literal");
      else if(lexeme[len]=='\'')
    {
     fprintf(fp,"<%s,%s>\n", "\"", "symbol");
        for (int i = 0; i < len + 1; i++) {
          lexeme[i] = lexeme[i + 1];
        }
        lexeme[len - 1] = '\0';
        fprintf(fp,"<%s,%s>\n", lexeme, "literal");
        fprintf(fp,"<%s,%s>\n", "\"", "symbol");
      }
      lexeme[0] = '\0';
      continue;
    }


    // checking if encountered a delimiter outside a string
literal
    L1:if (lit_flag == 0 && (isSpaces(curr_char) == 1 ||
isDelim(curr_char) == 1 ||
                          isOperator(curr_char) == 1)) {
      if (isKeyword(lexeme) == 1) {
        fprintf(fp,"<%s,%s>\n", lexeme, "keyword");
      } else if (isIdentifier(lexeme) == 1) {
        fprintf(fp,"<%s,%s>\n", lexeme, "identifier");
      } else if (isInteger(lexeme) == 1) {
        fprintf(fp,"<%s,%s>\n", lexeme, "integer");
      } else if (strlen(lexeme) > 0) {
        if (isOperator2(lexeme) == 1) {
          fprintf(fp,"<%s,%s>\n", lexeme, "operator");
        } else {
          fprintf(fp,"<%s,%s>\n", lexeme, "invalid identifier");
        }
      }
      if (isSpaces(curr_char) == 0) {
        if (isOperator(curr_char) == 1) {
          fprintf(fp,"<%c,%s>\n", curr_char, "operator");
        } else {
          fprintf(fp,"<%c,%s>\n", curr_char, "symbol");
        }
      }
      lexeme[0] = '\0';
    } else {
      // append to lexeme until a delimiter is encountered
      int len = strlen(lexeme);
      lexeme[len] = curr_char;
      lexeme[len + 1] = '\0';
    }
  }
}

int main() {
  FILE *f1;
  f1 = fopen(file, "r");
  fseek(f1, 0, SEEK_SET);
  fp=fopen("lex.txt","w");

  int c=1;
  char line[1024];
  while (fgets(line, 1024, f1)) {
    fprintf(fp,"\n%d. ",c++);
    handle_line(line);
  };
  // fclose(file);
  return 0;
}
```

fucntions.h

```
#define KW_LEN 32
const char *keywords[] = {
    "auto",   "break",  "case",     "char",    "continue", "do",
"default",
    "const",  "double", "else",     "enum",    "extern",   "for",
"if",
    "goto",   "float",  "int",      "long",    "register",
"return",  "signed",
    "static", "sizeof", "short",    "struct",  "switch",
"typedef", "union",
    "void",   "while",  "volatile", "unsigned", "FILE"};

int isKeyword(char *lexeme) {
  for (int i = 0; i < KW_LEN; i++) {
    if (strncmp(lexeme, keywords[i], 10) == 0) return 1;
  }
  return 0;
}

#define OP_LEN 11
const char operators[] = {'-', '+', '/', '*', '#', '=',
                          '&', '!', '|', '^', '%', '\0'};

int isOperator(char lexeme) {
  for (int i = 0; i < sizeof(operators); i++) {
    if (lexeme == operators[i]) return 1;
  }
  return 0;
}

#define OP_LEN2 6
const char *operators2[] = {"&&", "||", "==", ">=", "<=", "!=",
"-", "+", "/",
                            "*",  "#",  "=",  "&",  "!",  "|",
"^", "%"};

int isOperator2(char *lexeme) {
  int len = strlen(lexeme);
  if (len != 2) return 0;
  for (int i = 0; i < OP_LEN2; i++) {
    if (lexeme[0] == operators2[i][0] && lexeme[1] ==
operators2[i][1])
      return 1;
  }
  return 0;
}

int isIdentifier(char *lexeme) {
  if (isdigit(lexeme[0]) || strlen(lexeme) < 1) {
    return 0;
  }
  for (int i = 1; i < strlen(lexeme); i++) {
    if (!isalpha(lexeme[i]) && !isdigit(lexeme[i]) && lexeme[i] !=
'_')
      return 0;
  }
  return 1;
}

int isInteger(char *lexeme) {
  if (strlen(lexeme) < 1) return 0;

  for (int i = 0; i < strlen(lexeme); i++) {
    if (lexeme[i] < '0' || lexeme[i] > '9') return 0;
  }
  return 1;
}

#define DEL_LEN 10
const char delimeters[] = {'{', '}', '[', ']', '(',  ')',
                           '<', '>', ';', ',', '\0', '\n'};

int isDelim(char lexeme) {
  for (int i = 0; i < DEL_LEN; i++) {
    if (lexeme == delimeters[i]) return 1;
  }
  return 0;
}

int isSpaces(char lexeme) {
  if (lexeme == ' ' || lexeme == '\n' || lexeme == '\t')
    return 1;
  else
    return 0;
}
```

Output

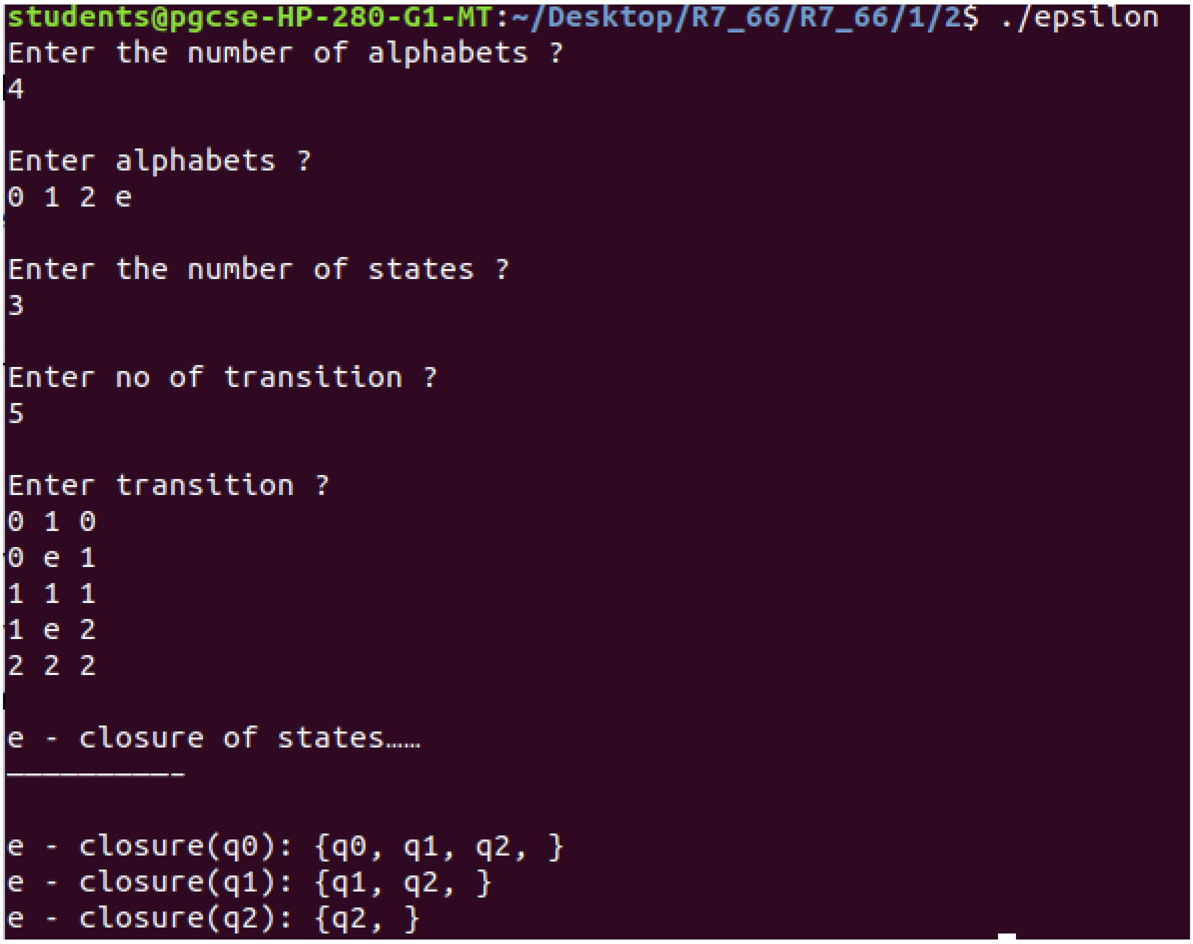
```
1. <#,operator>
<include,identifier>
<<,symbol>
<ctype.h,invalid identifier>
<>,symbol>

2. <#,operator>
<include,identifier>
<<,symbol>
<stdio.h,invalid identifier>
<>,symbol>

3. <#,operator>
<include,identifier>
<<,symbol>
<stdlib.h,invalid identifier>
<>,symbol>

4. <#,operator>
<include,identifier>
<<,symbol>
<string.h,invalid identifier>
<>,symbol>

5.
6. <#,operator>
<include,identifier>
<",symbol>
<functions.h,literal>
<",symbol>

7.
8. <#,operator>
<define,identifier>
<file,identifier>
<",symbol>
<input.c,literal>
<",symbol>

9.
10. <#,operator>
<define,identifier>
<debug,identifier>
<(,symbol>
<x,identifier>
<),symbol>
<puts,identifier>
<(,symbol>
<x,identifier>
<),symbol>

11. <static,keyword>
<int,keyword>
<comment_flag,identifier>
<=,operator>
<0,integer>
<;,symbol>
```

```
12. Single line comment
13.
14. multiline comment
15. multiline comment
16.
                                  .
                                  .
                                  .
                                  .
                                  .
                                  .
                                  .

136. <while,keyword>
<(,symbol>
<fgets,identifier>
<(,symbol>
<line,identifier>
<,,symbol>
<1024,integer>
<,,symbol>
<f1,identifier>
<),symbol>
<),symbol>
<{,symbol>

137. <fprintf,identifier>
<(,symbol>
<fp,identifier>
<,,symbol>
<",symbol>
<\n%d. ,literal>
<",symbol>
<,,symbol>
<c,identifier>
<+,operator>
<+,operator>
<),symbol>
<;,symbol>

138. <handle_line,identifier>
<(,symbol>
<line,identifier>
<),symbol>
<;,symbol>

139. <},symbol>
<;,symbol>

140. Single line comment
141. <return,keyword>
<0,integer>
<;,symbol>

142. <},symbol>
```

Program Code

```
#include<stdio.h>

#include<stdlib.h>

struct node {
  int st;
  struct node *link;
};

int state, alpha, s, no_transition,c,r, buffer[20];
char alphabet[20];

int e_closure[20][20] = { 0 };
struct node *transition[20][20] = { NULL };

void findclosure(int x, int sta)
{
  struct node * temp;
  int i;
  if (buffer[x])
    return;
  e_closure[sta][c++] = x;
  buffer[x] = 1;
  if (alphabet[alpha - 1] == 'e' && transition[x][alpha - 1] !=
NULL)
  {
    temp = transition[x][alpha - 1];
    while (temp != NULL)
    {
      findclosure(temp->st, sta);
      temp = temp -> link;
    }
  }
}


int findalpha(char y)
{
  for (int i = 0; i < alpha; i++)
  {
    if (alphabet[i] == y)
    {
      return i;
    }
  }
  return (999);
}

void insert_trantbl(int r, char y, int s)
{
  int j;
  struct node * temp;
  j = findalpha(y);
  if (j == 999)
  {
    printf("error\n");
    exit(0);
  }
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->st = s;
  temp->link = transition[r][j];
  transition[r][j] = temp;
}

void print_e_closure(int i)
{
  int j;
  printf("{");
    for (j = 0; e_closure[i][j] != -1; j++)
      printf("q%d, ",e_closure[i][j]);
     printf("}");
}

void main() {
  int i, j, k, m, t, n;
  char y;

  struct node *temp;
  printf("Enter the number of alphabets ? \n");
  scanf(" %d", &alpha);
  printf("\nEnter alphabets ? \n");
  for (i = 0; i < alpha; i++)
  {
    scanf(" %c",&alphabet[i]);
  }
  printf("\nEnter the number of states ? \n");
  scanf("%d", &state);
  printf("\nEnter no of transition ? \n");
  scanf("%d", &no_transition);
  printf("\nEnter transition ? \n");
  for (i = 0; i < no_transition; i++)
  {
    scanf("%d %c%d",&r,&y,&s);
    insert_trantbl(r,y,s);
  }
  printf("\n");
  printf("e - closure of states……\n");
  printf("—————————-\n");

  for (i = 0; i<state; i++)
  {
    c = 0;
    for (j = 0; j < 20; j++)
    {
      buffer[j] = 0;
      e_closure[i][j] = -1;
    }
    findclosure(i, i);
    printf("\ne - closure(q%d): ", i);
    print_e_closure(i);

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

<u>**Output**</u>



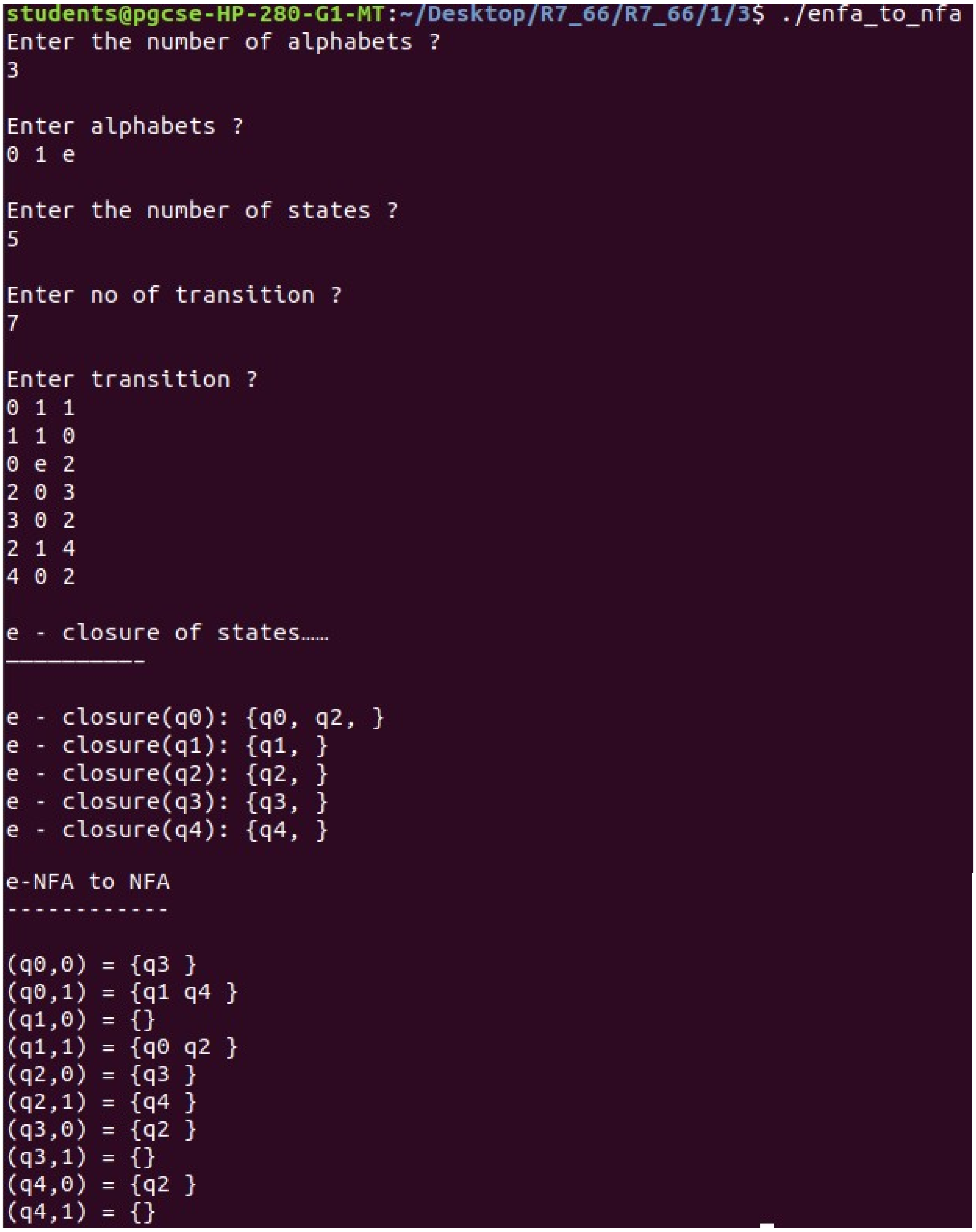
```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/1/2$ ./epsilon
Enter the number of alphabets ?
4

Enter alphabets ?
0 1 2 e

Enter the number of states ?
3

Enter no of transition ?
5

Enter transition ?
0 1 0
0 e 1
1 1 1
1 e 2
2 2 2

e - closure of states……
-----------

e - closure(q0): {q0, q1, q2, }
e - closure(q1): {q1, q2, }
e - closure(q2): {q2, }
```

Program Code

```
#include<stdio.h>

#include<stdlib.h>

struct node {
  int st;
  struct node *link;
};

int state, alpha, s, no_transition,c,r, buffer[20];
char alphabet[20];

int e_closure[20][20] = { 0 };
struct node *transition[20][20] = { NULL };

void print_e_closure(int i)
{
  int j;
  printf("{");
    for (j = 0; e_closure[i][j] != -1; j++)
      printf("q%d, ",e_closure[i][j]);
     printf("}");
}


int findalpha(char y)
{
  for (int i = 0; i < alpha; i++)
  {
    if (alphabet[i] == y)
    {
      return i;
    }
  }
  return (999);
}

void add_closure(int k, int visited[],int t)
{
  for (int j = 0; e_closure[k][j] != -1; j++)
  {
      int flag=0;
      for(int i=0;i<t;i++)
      {
        if(e_closure[k][j]==visited[i])
        {
          flag=1;
          break;
        }
      }
      if(!flag)
      {
        visited[t++]=e_closure[k][j];
        printf("q%d ",visited[t-1]);
      }
    }
}
```

```
void enfa_to_nfa(int i,char y)
{
  int t=0;
  int visited[20];
  if(y!='e')
  {
    printf("\n(q%d,%c) = {",i,y);
    for (int j = 0; e_closure[i][j] != -1; j++)
    {
      int u=findalpha(y);
      if (u == 999)
      {
        printf("error\n");
        exit(0);
      }
      if(transition[e_closure[i][j]][u]!=NULL)
      {
        int k=transition[e_closure[i][j]][u]->st;
        add_closure(k,visited,t);
      }
    }
    printf("}");
  }
}

void findclosure(int x, int sta)
{
  struct node * temp;
  int i;
  if (buffer[x])
    return;
  e_closure[sta][c++] = x;
  buffer[x] = 1;
  if (alphabet[alpha - 1] == 'e' && transition[x][alpha - 1] !=
NULL)
  {
    temp = transition[x][alpha - 1];
    while (temp != NULL)
    {
      findclosure(temp->st, sta);
      temp = temp -> link;
    }
  }
}


void insert_trantbl(int r, char y, int s)
{
  int j;
  struct node * temp;
  j = findalpha(y);
  if (j == 999)
  {
    printf("error\n");
    exit(0);
  }
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->st = s;
  temp->link = transition[r][j];
  transition[r][j] = temp;
}



void main() {
  int i, j, k, m, t, n;
  char y;

  struct node *temp;
  printf("Enter the number of alphabets ? \n");
  scanf(" %d", &alpha);
  printf("\nEnter alphabets ? \n");
  for (i = 0; i < alpha; i++)
  {
    scanf(" %c",&alphabet[i]);
  }
  printf("\nEnter the number of states ? \n");
  scanf("%d", &state);
  printf("\nEnter no of transition ? \n");
  scanf("%d", &no_transition);
  printf("\nEnter transition ? \n");
  for (i = 0; i < no_transition; i++)
  {
    scanf("%d %c%d",&r,&y,&s);
    insert_trantbl(r,y,s);
  }
  printf("\n");
  printf("e - closure of states……\n");
  printf("——————————-\n");

  for (i = 0; i<state; i++)
  {
    c = 0;
    for (j = 0; j < 20; j++)
    {
      buffer[j] = 0;
      e_closure[i][j] = -1;
    }
    findclosure(i, i);
    printf("\ne - closure(q%d): ", i);
    print_e_closure(i);
  }

  printf("\n\ne-NFA to NFA");
  printf("\n-----------\n");
  for(int i=0;i<state;i++)
  {
    for(int j=0;j<alpha;j++)
    {
      enfa_to_nfa(i,alphabet[j]);
    }
  }
  printf("\n");
}
```

Output



```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/1/3$ ./enfa_to_nfa
Enter the number of alphabets ?
3

Enter alphabets ?
0 1 e

Enter the number of states ?
5

Enter no of transition ?
7

Enter transition ?
0 1 1
1 1 0
0 e 2
2 0 3
3 0 2
2 1 4
4 0 2

e - closure of states……
__________-

e - closure(q0): {q0, q2, }
e - closure(q1): {q1, }
e - closure(q2): {q2, }
e - closure(q3): {q3, }
e - closure(q4): {q4, }

e-NFA to NFA
------------

(q0,0) = {q3 }
(q0,1) = {q1 q4 }
(q1,0) = {}
(q1,1) = {q0 q2 }
(q2,0) = {q3 }
(q2,1) = {q4 }
(q3,0) = {q2 }
(q3,1) = {}
(q4,0) = {q2 }
(q4,1) = {}
```

Program Code

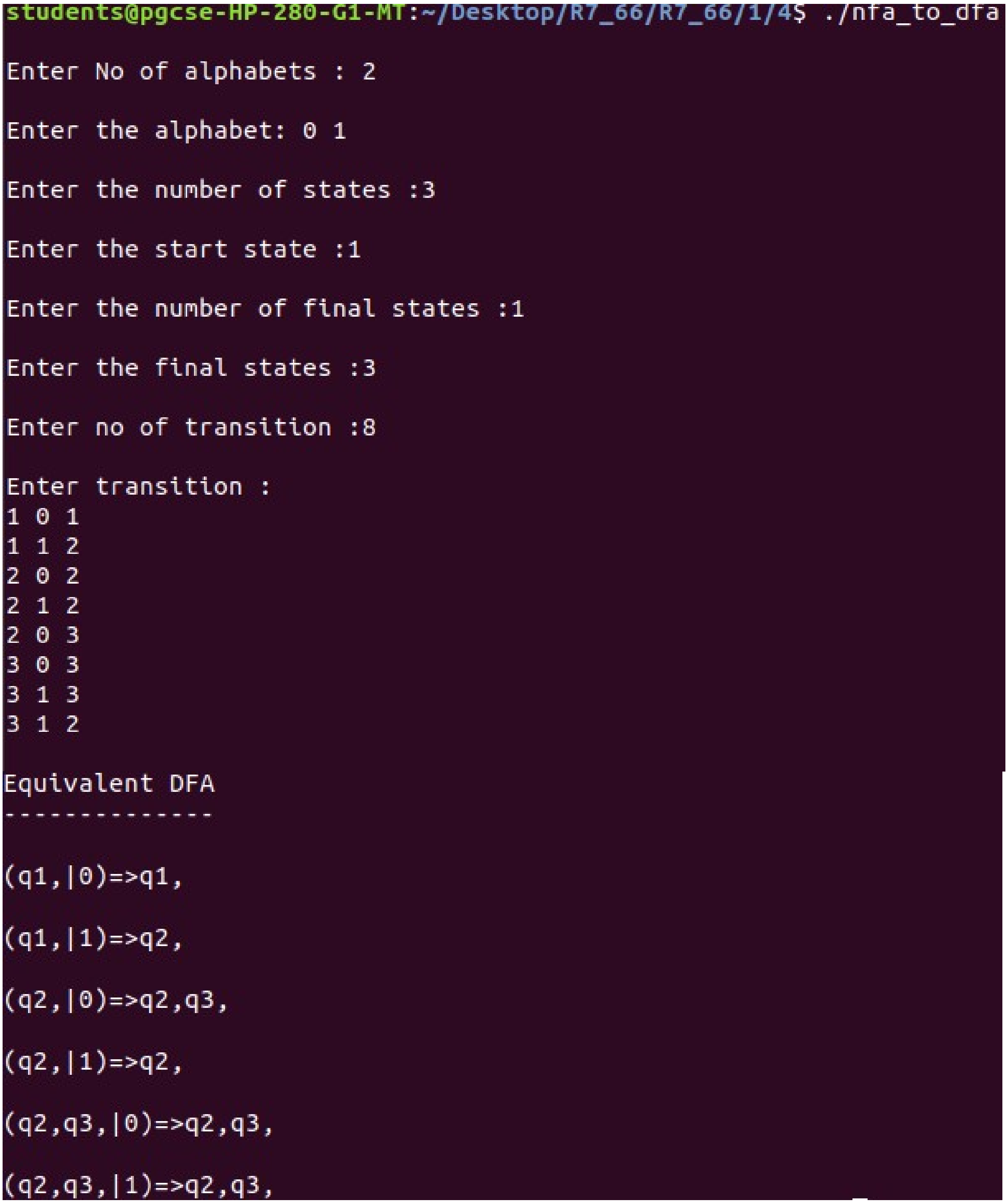
```
#include<stdio.h>

#include<stdlib.h>

struct node {
  int st;
  struct node * link;
};
struct node1 {
  int nst[20];
};

int set[20], nostate, noalpha, s, notransition, nofinal, start,
finalstate[20], c, r, buffer[20];
int complete = -1;
char alphabet[20];
static int eclosure[20][20] = {0};
struct node1 hash[20];
struct node * transition[20][20] = {NULL};

int compare(struct node1 a, struct node1 b)
{
  int i;

  for (i = 1; i <= nostate; i++)
  {
    if (a.nst[i] != b.nst[i])
      return 0;
  }
  return 1;
}

int insert_dfa_state(struct node1 newstate)
{
  int i;
  for (i = 0; i <= complete; i++)
  {
    //checking if the state was already added
    if (compare(hash[i], newstate))
      return 0;
  }
  complete++;

  //marking the new state as completed
  hash[complete] = newstate;
  return 1;
}

int findalpha(char c)
{
  int i;
  for (i = 0; i < noalpha; i++)
    if (alphabet[i] == c)
      return i;

  return (999);
}
```

```c
void insert(int r, char c, int s)
{
  int j;
  struct node * temp;
  j = findalpha(c);
  if (j == 999)
  {
    printf("error\n");
    exit(0);
  }
  temp = (struct node * ) malloc(sizeof(struct node));
  temp -> st = s;
  temp -> link = transition[r][j];
  transition[r][j] = temp;
}

void printnewstate(struct node1 state)
{
  int j;
  for (j = 1; j <= nostate; j++)
  {
    if (state.nst[j] != 0)
      printf("q%d,", state.nst[j]);
  }

}

void main()
{
  int i, j, k, m, t, n, l;
  struct node * temp;
  struct node1 newstate = {0},tmpstate = {0};

  printf("\nEnter No of alphabets : ");
  scanf("%d",&noalpha);
  printf("\nEnter the alphabet: ");
  for (i = 0; i < noalpha; i++)
  {
    scanf(" %c",&alphabet[i]);
  }
  printf("\nEnter the number of states :");
  scanf("%d", & nostate);
  printf("\nEnter the start state :");
  scanf("%d", & start);
  printf("\nEnter the number of final states :");
  scanf("%d", & nofinal);
  printf("\nEnter the final states :");
  for (i = 0; i < nofinal; i++)
    scanf("%d",&finalstate[i]);
  printf("\nEnter no of transition :");

  scanf("%d", & notransition);

  printf("\nEnter transition :");


  //Create the transition table

  char y;
  for (i = 0; i < notransition; i++)
```

```
{
  scanf("%d %c%d", & r, & y, & s);
  insert(r, y, s);
}

for (i = 0; i < 20; i++)
{
  for (j = 0; j < 20; j++)
    hash[i].nst[j] = 0;
}
complete = -1;
i = -1;
printf("\nEquivalent DFA\n");
printf("--------------\n");

//adding initial state
newstate.nst[start] = start;
insert_dfa_state(newstate);
while (i != complete)
{
  i++;
  newstate = hash[i];
  for (k = 0; k < noalpha; k++)
  {
    c = 0;
    for (j = 1; j <= nostate; j++)
      set[j] = 0;
    for (j = 1; j <= nostate; j++)
    {
      l = newstate.nst[j];
      if (l != 0)
      {
        temp = transition[l][k];
        while (temp != NULL)
        {
          if (set[temp -> st] == 0)
          {
            c++;
            set[temp -> st] = temp -> st;
          }
          temp = temp -> link;

        }
      }
    }
    printf("\n");
    if (c != 0)
    {
      for (m = 1; m <= nostate; m++)
        tmpstate.nst[m] = set[m];

      insert_dfa_state(tmpstate);
      printf("(");
      printnewstate(newstate);
      printf("|%c", alphabet[k]);
      printf(")=>");
      printnewstate(tmpstate);
      printf("\n");
    }
    else
```

```
      {
        printf("(");
        printnewstate(newstate);
        printf("|%c", alphabet[k]);
        printf(")=>");
        printf("NULL\n");
      }

    }
  }
}
```

Output



```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/1/4$ ./nfa_to_dfa

Enter No of alphabets : 2

Enter the alphabet: 0 1

Enter the number of states :3

Enter the start state :1

Enter the number of final states :1

Enter the final states :3

Enter no of transition :8

Enter transition :
1 0 1
1 1 2
2 0 2
2 1 2
2 0 3
3 0 3
3 1 3
3 1 2

Equivalent DFA
--------------

(q1,|0)=>q1,

(q1,|1)=>q2,

(q2,|0)=>q2,q3,

(q2,|1)=>q2,

(q2,q3,|0)=>q2,q3,

(q2,q3,|1)=>q2,q3,
```

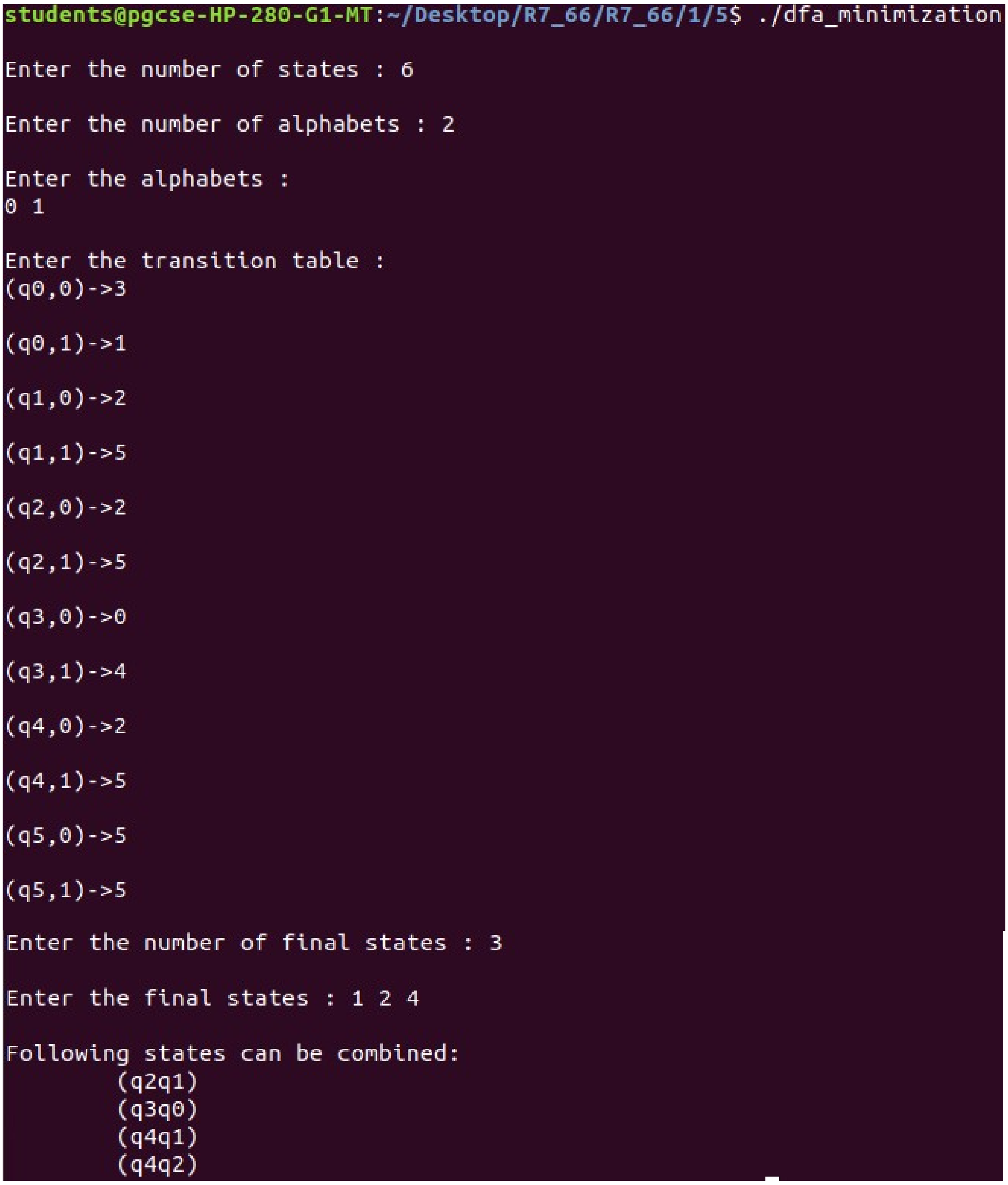
Program Code

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


int state, alpha, k;

int check(int p,int unvisited[50])
{
     for(int i=0;i<k;i++)
     {
          if(p==unvisited[i])
               return i;
     }
     return -1;
}

void main()
{

     printf("\nEnter the number of states : ");
     scanf("%d",&state);

     printf("\nEnter the number of alphabets : ");
     scanf("%d",&alpha);

     int alphabet[alpha];

     printf("\nEnter the alphabets : \n");
     for(int i=0;i<alpha;i++)
          scanf("%d",&alphabet[i]);

     int transition[state][alpha];

     printf("\nEnter the transition table : ");

     for(int i=0;i<state;i++)
          for(int j=0;j<alpha;j++)
          {
               printf("\n(q%d,%d)->",i,alphabet[j]);
               scanf("%d",&transition[i][j]);
          }

     int myhill[state][state];

     for(int i=0;i<state;i++)
     {
          for(int j=0;j<state;j++)
               myhill[i][j]=-1;
     }

     printf("\nEnter the number of final states : ");
     int final;
     scanf("%d",&final);

     int fin[final];

     printf("\nEnter the final states : ");
```

```
for(int i=0;i<final;i++)
     scanf("%d",&fin[i]);

for(int i=0;i<final;i++)
{
     for(int j=0;j<state;j++)
     {
          int flag=0;
          for(int k=0;k<final;k++)
          {
               if(j==fin[k])
               {
                    flag=1;
                    break;
               }
          }
          if(flag==0)
          {
               myhill[j][fin[i]] = 1;
               myhill[fin[i]][j] = 1;
          }

     }
}
int c;
do
{
     c=0;
     for(int i=1;i<state;i++)
     {
          for(int j=0;j<i;j++)
          {
               if(myhill[i][j]==-1)
               {
                    for(int k=0;k<alpha;k++)
                    {
                         int a = transition[i][k];
                         int b = transition[j][k];
                         //printf ("\n%d %d %d %d",a,b,i,j);
                         if(myhill[a][b]!=-1)
                         {
                              myhill[i][j]=1;
                              c++;
                              break;
                         }
                    }
               }
          }
     }
}while(c>0);

printf("\nFollowing states can be combined: ");

for(int i=1;i<state;i++)
{
     for(int j=0;j<i;j++)
     {
          if(myhill[i][j]==-1)
          {
               printf("\n\t(q%dq%d)",i,j);
```

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

    }
```

Output



```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/1/5$ ./dfa_minimization

Enter the number of states : 6

Enter the number of alphabets : 2

Enter the alphabets :
0 1

Enter the transition table :
(q0,0)->3

(q0,1)->1

(q1,0)->2

(q1,1)->5

(q2,0)->2

(q2,1)->5

(q3,0)->0

(q3,1)->4

(q4,0)->2

(q4,1)->5

(q5,0)->5

(q5,1)->5

Enter the number of final states : 3

Enter the final states : 1 2 4

Following states can be combined:
        (q2q1)
        (q3q0)
        (q4q1)
        (q4q2)
```

Program Code

lex.l

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

    int comment=0;
%}

operator [+*\-&/%=!#\[\]]|(<=)|(>=)|[<>]
letter [a-zA-Z]
digit [0-9]
literal (["]({letter}|{operator}|{digit}|[\\n]|[\\t]|[ ])*["])|
{digit}+
identifier ({letter}|_)({letter}|_|{digit})*

%%
\/\/.*   ;
\/\*.* ;
.*\*\/.*  ;
void|main|include|define|printf|scanf|fgets|for|while|int|char|
strlen|FILE|fopen|feof|NULL|if|return|double|continue|break|
strcmp|strcat|fflush|fscanf|fprintf|strcpy|return {printf("\n%s,
keyword",yytext);}
{operator} {printf("\n%s, operator",yytext);}
"1","2","3","4","5","6","7","8","9","0" {printf("\n%s,
literal",yytext);}
{literal} {printf("\n%s, literal",yytext);}
"{"|"}"|"("|")"|";"|","|"." {printf("\n%s, seperator",yytext);}
{identifier} {printf("\n%s, identifier",yytext);}
%%

int yywrap(){}
void main()
{
    yyin=fopen("test.c", "r");
    yylex();
}
```

Test.c

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

/*This is an implementation of lexical analyser using the lex
tool. This program was implemented for the compiler lab*/

void main()
{
    int a,b,c;
    a=5;
    b=8;
    c=a+b*a;
    printf("\nc = %d\n",c);
}
```

Output

```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/1$ lex
lexical.l
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/1$ cc lex.yy.c
-o lex
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/1$ ./lex

#, operator
include, keyword
<, operator
stdio, identifier
., seperator
h, identifier
>, operator

#, operator
include, keyword
<, operator
stdlib, identifier
., seperator
h, identifier
>, operator




void, keyword
main, keyword
(, seperator
), seperator

{, seperator

int, keyword
a, identifier
,, seperator
b, identifier
,, seperator
c, identifier
;, seperator

a, identifier
=, operator
5, literal
;, seperator

b, identifier
=, operator
8, literal
;, seperator

c, identifier
=, operator
a, identifier
+, operator
b, identifier
*, operator
a, identifier
;, seperator
printf, keyword
(, seperator
"\nc = %d\n", literal
,, seperator
c, identifier
), seperator
;, seperator

}, seperator
```

Program Code

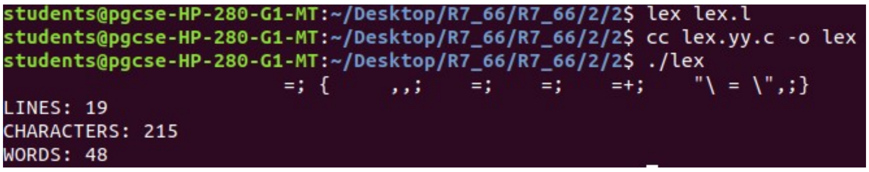
lex.l

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

    int l=0;
    int w=0;
    int c=0;
%}

line [\n]
words [a-zA-Z0-9!@#$%^&*<>/?.()]+

%%
{line} {l++;}
{words} {w++, c+=strlen(yytext);}
%%

int yywrap(){}

void main()
{
    yyin=fopen("test.c", "r");
    yylex();

    printf("\nLINES: %d", l+1);
    printf("\nCHARACTERS: %d",c);
    printf("\nWORDS: %d\n", w);
}
```

test.c

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


//This program was created by sriganash

/*This is an implementation of lexical analyser using the lex
tool. This program was implemented for the compiler lab*/
int a=10;

void main()
{
    int a,b,c;
    a=5;
    b=8;
    c=a+b*a;
    printf("\nc = %d\n",c);
}
```

Output



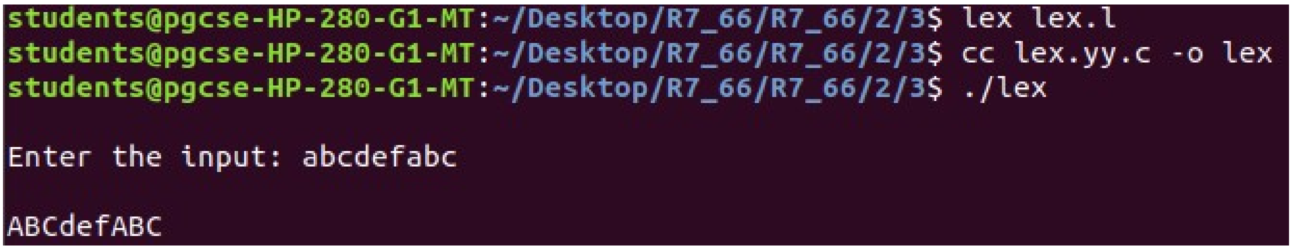
```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/2$ lex lex.l
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/2$ cc lex.yy.c -o lex
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/2$ ./lex
                      =; {    ,,;   =;   =;    =+;   "\ = \",;}
LINES: 19
CHARACTERS: 215
WORDS: 48
```

Program Code

lex.l

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

%}

string ([d-z]*(abc)*[d-z]*)*

%%
{string} {
            int n=strlen(yytext);
            for(int i=0;i<n-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("\n%s\n",yytext);
         }

%%

int yywrap(){}

void main()
{
    printf("\nEnter the input: ");
    yylex();

}
```

Output

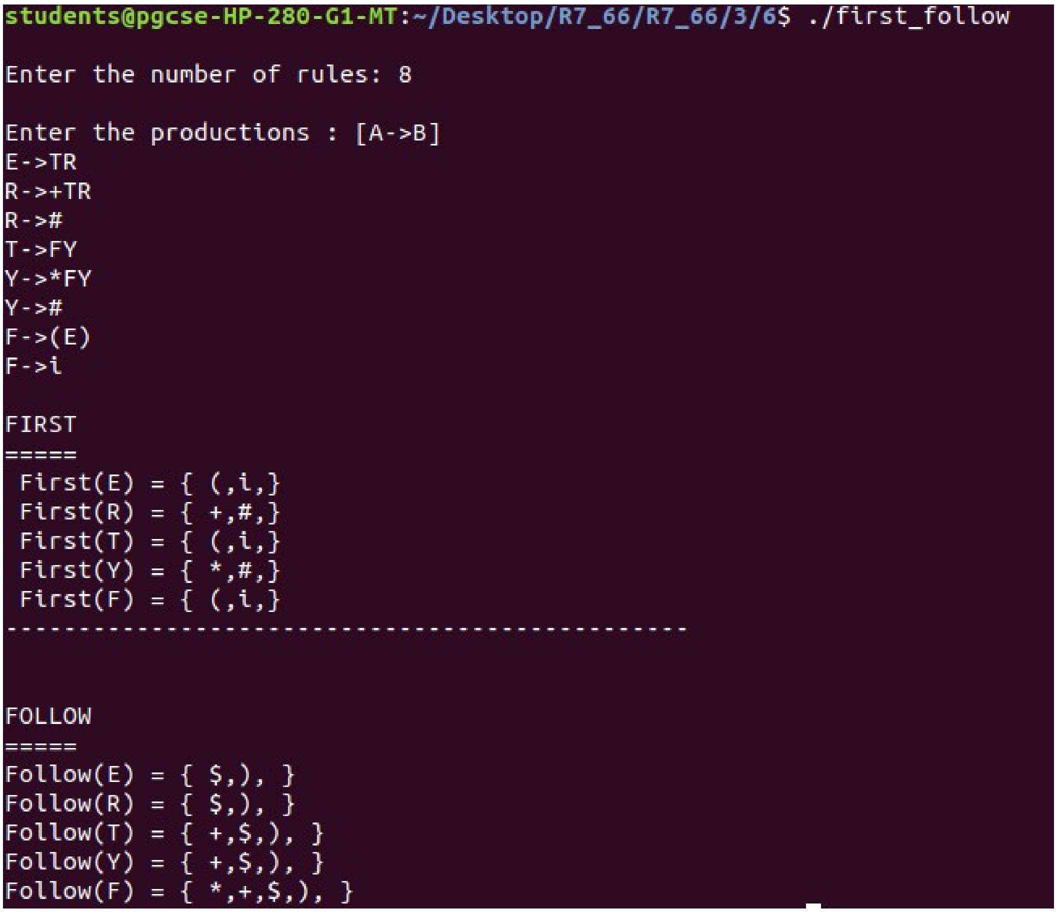


```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/3$ lex lex.l
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/3$ cc lex.yy.c -o lex
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/2/3$ ./lex

Enter the input: abcdefabc

ABCdefABC
```

Program Code

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

void followfirst(char, int, int);
void follow(char c);

void findfirst(char, int, int);

int count, n = 0;

char calc_first[10][100];

char calc_follow[10][100];
int m = 0;

char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

void main()
{
    int jm = 0;
    int km = 0;
    int i, choice;
    char c, ch;

    printf("\nEnter the number of rules: ");
    scanf("%d",&count);

    printf("\nEnter the productions : [A->B]");

    for(int i=0;i<count;i++)
    {
        scanf(" %s",production[i]);
    }

    char done[count];
    int ptr = -1;

    for(k = 0; k < count; k++)
    {
        for(int z = 0; z < 100; z++)
        {
            calc_first[k][z] = '!';
        }
    }
    int point1 = 0, point2, x;

    printf("\nFIRST");
    printf("\n=====");
    for(k = 0; k < count; k++)
    {
        c = production[k][0];
        point2 = 0;
        x = 0;
        for(int z = 0; z <= ptr; z++)
        {
            if(c == done[z])
                x = 1;
        }

        if (x == 1)
            continue;

        //Finding the First of the terminal of current production
        findfirst(c, 0, 0);

        done[++ptr] = c;
        printf("\n First(%c) = { ", c);
        calc_first[point1][point2++] = c;

        for(i = 0 + jm; i < n; i++)
        {
            int chk = 0;
            for(int l = 0; l < point2; l++)
            {
                 //checking if there is any repetition
                if (first[i] == calc_first[point1][l])
                {
                    chk = 1;
                    break;
                }
            }
            if(chk == 0)
            {
                printf("%c,", first[i]);
                calc_first[point1][point2++] = first[i];
            }
        }
        printf("}");
        jm = n;
        point1++;
    }
    printf("\n-----------------------------------------------\n\
n");
    printf("\nFOLLOW");
    printf("\n=====");
    char donee[count];
    ptr = -1;

    for(k = 0; k < count; k++)
    {
        for(int z = 0; z < 100; z++)
        {
            calc_follow[k][z] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for(e = 0; e < count; e++)
    {
        ck = production[e][0];
        point2 = 0;
        x = 0;
        for(int z = 0; z <= ptr; z++)
            if(ck == donee[z])
                x = 1;

        if (x == 1)
            continue;
        land += 1;

        follow(ck);

        donee[++ptr] = ck;
        printf("\nFollow(%c) = { ", ck);
        calc_follow[point1][point2++] = ck;

        for(i = 0 + km; i < m; i++)
        {
            int l = 0, chk = 0;
            for(l = 0; l < point2; l++)
            {
                if (f[i] == calc_follow[point1][l])
                {
                    chk = 1;
                    break;
                }
            }
            if(chk == 0)
            {
                printf("%c,", f[i]);
                calc_follow[point1][point2++] = f[i];
            }
        }
        printf(" }");
        km = m;
        point1++;
    }
    printf("\n");
}

void follow(char c)
{
    int i, j;

    if(production[0][0] == c)
    {
        f[m++] = '$';
    }
    for(i = 0; i < 10; i++)
    {
        for(j = 3;j < 10; j++)
        {
            if(production[i][j] == c)
            {
                if(production[i][j+1] != '\0')
                {
                    //if not the end of the production call
followfirst()
                    followfirst(production[i][j+1], i, (j+2));
                }
                if(production[i][j+1]=='\0' && c!=production[i]
[0])
                {
                    //else call follow() of the current non-
terminal of the production
                    follow(production[i][0]);
                }
            }
        }
    }
}

void findfirst(char c, int q1, int q2)
{
    int j;

    if(!(isupper(c)))
    {
        first[n++] = c;
    }
    for(j = 0; j < count; j++)
    {
        if(production[j][0] == c)
        {
            //checking if the first of 'c' is epsilon
            if(production[j][3] == '#')
            {
                if(production[q1][q2] == '\0')
                    first[n++] = '#';
                else if(production[q1][q2] != '\0' && (q1 != 0 ||
q2 != 0))
                {
                    findfirst(production[q1][q2], q1, (q2+1));
                }
                else
                    first[n++] = '#';
            }
            else if(!isupper(production[j][3]))
            {
                //if terminal add terminal as first
                first[n++] = production[j][3];
            }
            else
            {
                //if non-terminal add call findfirst
                findfirst(production[j][3], j, 4);
            }
        }
    }
}

void followfirst(char c, int c1, int c2)
{
    int k;

    if(!(isupper(c)))
        f[m++] = c;
    else
    {
        int i = 0, j = 1;
        for(i = 0; i < count; i++)
        {
            if(calc_first[i][0] == c)
                break;
        }

        while(calc_first[i][j] != '!')
        {
            if(calc_first[i][j] != '#')
            {
                //if the first of the current non_terminal is not
epsiln then add it to f
                f[m++] = calc_first[i][j];
            }
            else
            {
                if(production[c1][c2] == '\0')
                {
                    //if the end of the production is reached call
follow() of the non_terminal of the current production
                    follow(production[c1][0]);
                }
                else
                {
                    //else call followfirst() of the current
productions next symbol
                    followfirst(production[c1][c2], c1, c2+1);
                }
            }
            j++;
        }
    }
}
```

Output

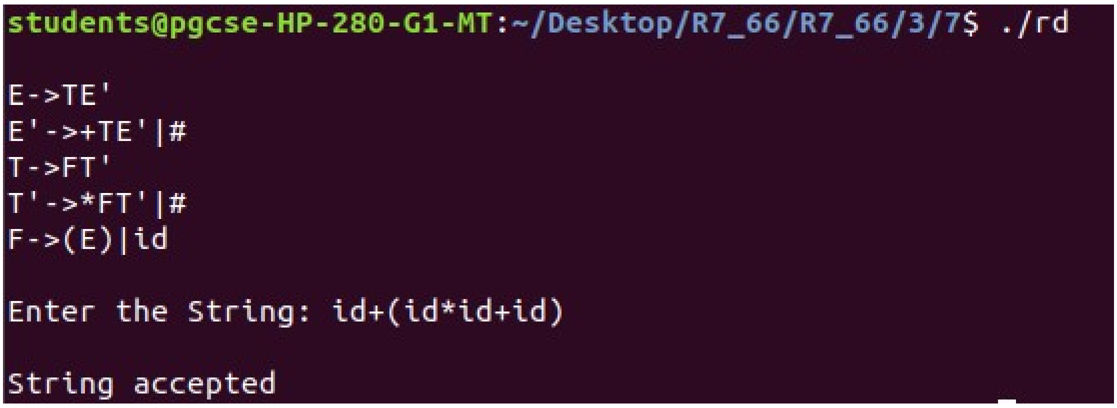


```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/3/6$ ./first_follow

Enter the number of rules: 8

Enter the productions : [A->B]
E->TR
R->+TR
R->#
T->FY
Y->*FY
Y->#
F->(E)
F->i

FIRST
=====
 First(E) = { (,i,}
 First(R) = { +,#,}
 First(T) = { (,i,}
 First(Y) = { *,#,}
 First(F) = { (,i,}
-----------------------------------------------


FOLLOW
=====
Follow(E) = { $,), }
Follow(R) = { $,), }
Follow(T) = { +,$,), }
Follow(Y) = { +,$,), }
Follow(F) = { *,+,$,), }
```

Program Code

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

char production[20][20];
int count;
char string[20];
int ptr;

int E();
int E_();
int T();
int T_();
int F();

void main()
{
    ptr=0;
    count=6;
    strcpy(production[0],"E->E+T|T");
    strcpy(production[1],"T->T*F|F");
    strcpy(production[2],"F->(E)|id");

    for(int i=0;i<3;i++)
    {
        char c=production[i][0];

        if(c==production[i][3])
        {
            printf("\n%c->",c);
            int p=-1;
            for(int j=4;j<strlen(production[i]);j++)
            {
                if(production[i][j]=='|')
                {
                    p=j;
                    break;
                }
            }
            if(p!=-1)
            {
                for(int k=p+1;k<strlen(production[i]);k++)
                {
                    printf("%c",production[i][k]);
                }
            }
            printf("%c'",c);
            printf("\n%c'->",c);
            for(int j=4;production[i][j]!='|';j++)
            {
                printf("%c",production[i][j]);
            }
            printf("%c'|#",c);
        }
        else
        {
            printf("\n%s",production[i]);
        }
    }
    printf("\n");

    printf("\nEnter the String: ");
    scanf(" %s",string);

    if(E() && ptr==strlen(string))
    {
        printf("\nString accepted\n");
    }
    else
    {
        printf("\nString not accepted\n");
    }
}

int F()
{
    if(string[ptr]=='(')
    {
        ptr++;
        if(E())
        {
            if(string[ptr]==')')
            {
                ptr++;
                return 1;
            }
            return 0;
        }
        return 0;
    }
    else if(string[ptr]=='i')
    {
        ptr++;
        if(string[ptr]=='d')
        {
            ptr++;
            return 1;
        }
        return 0;
    }
    return 0;
}

int T_()
{
    if(string[ptr]=='*')
    {
        ptr++;
        if(F())
        {
            if(T_())
            {
                return 1;
            }
            return 0;
        }
        return 0;
    }
    return 1;
}

int T()
{
    if(F())
    {
        if(T_())
        {
            return 1;
        }
        return 0;
    }
    return 0;
}

int E_()
{
    if(string[ptr]=='+')
    {
        ptr++;
        if(T())
        {
            if(E_())
            {
                return 1;
            }
            return 0;
        }
        return 0;
    }
    return 1;
}

int E()
{
    if(T())
    {
        if(E_())
        {
            return 1;
        }
        return 0;
    }
    return 0;
}
```

Output

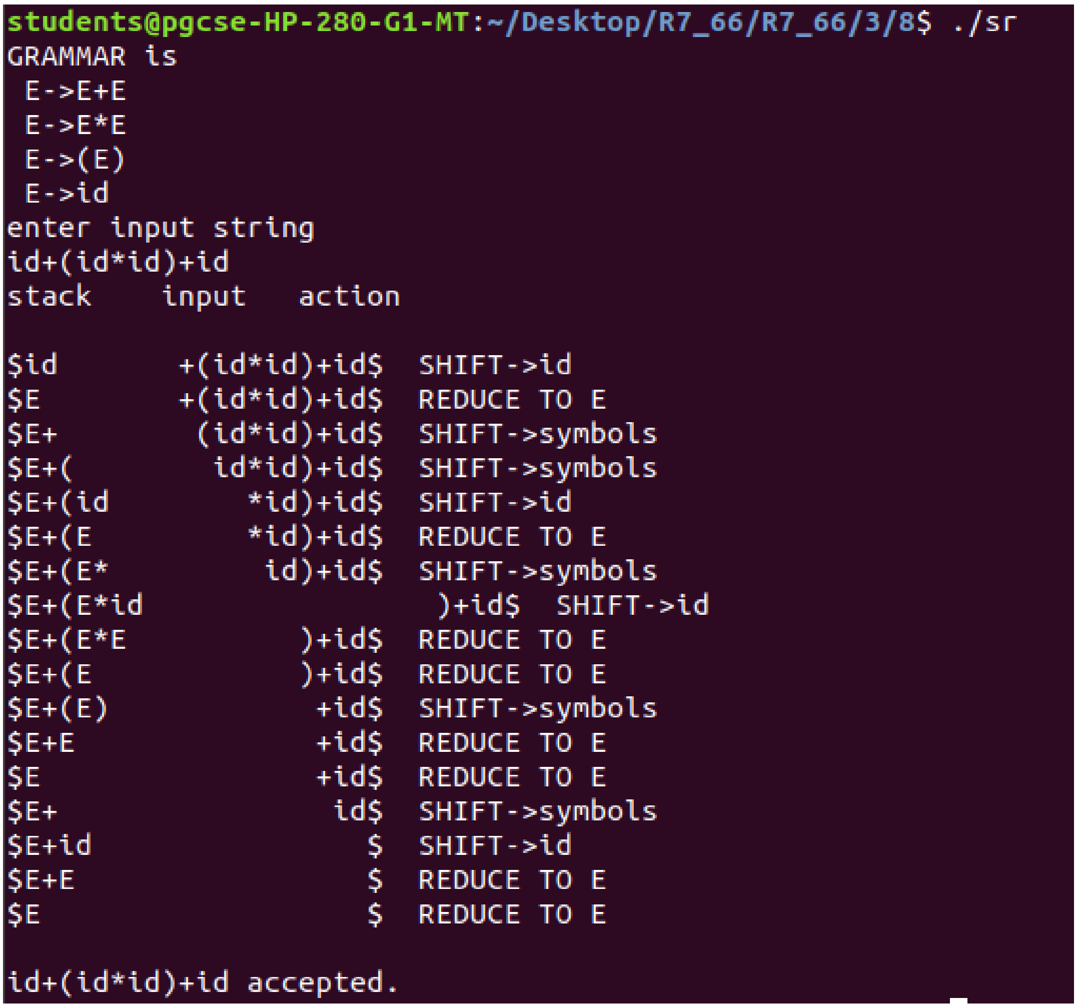


```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/3/7$ ./rd

E->TE'
E'->+TE'|#
T->FT'
T'->*FT'|#
F->(E)|id

Enter the String: id+(id*id+id)

String accepted
```

Program Code

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

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

 }


void main()
{
      puts("GRAMMAR is\n E->E+E \n E->E*E \n E->(E) \n E->id");
      puts("enter input string ");
      scanf("%s",a);
      c=strlen(a);
      char temp[50];
      strcpy(temp,a);
      strcpy(act,"SHIFT->");
      puts("stack \t input \t action");
      for(k=0,i=0; j<c; k++,i++,j++)
      {
         if(a[j]=='i' && a[j+1]=='d')
           {
              stk[i]=a[j];
              stk[i+1]=a[j+1];
              stk[i+2]='\0';
              a[j]=' ';
              a[j+1]=' ';
              printf("\n$%s\t%s$\t%sid",stk,a,act);
           }
         else
           {
              stk[i]=a[j];
              stk[i+1]='\0';
              a[j]=' ';
              printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
           }
           check();
       }

        printf("\n");
         if(strlen(stk)!=1)
         {
          printf("\n%s not accepted.\n",temp);
         }
         else
         {
          printf("\n%s accepted.\n",temp);
         }
}
```

Output



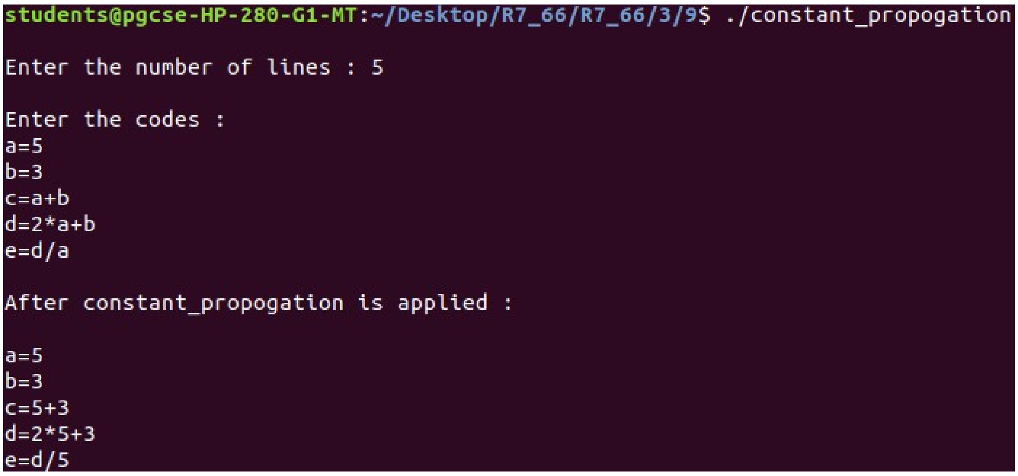
```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/3/8$ ./sr
GRAMMAR is
 E->E+E
 E->E*E
 E->(E)
 E->id
enter input string
id+(id*id)+id
stack    input   action

$id       +(id*id)+id$  SHIFT->id
$E        +(id*id)+id$  REDUCE TO E
$E+        (id*id)+id$  SHIFT->symbols
$E+(        id*id)+id$  SHIFT->symbols
$E+(id        *id)+id$  SHIFT->id
$E+(E         *id)+id$  REDUCE TO E
$E+(E*         id)+id$  SHIFT->symbols
$E+(E*id                 )+id$  SHIFT->id
$E+(E*E          )+id$  REDUCE TO E
$E+(E            )+id$  REDUCE TO E
$E+(E)            +id$  SHIFT->symbols
$E+E              +id$  REDUCE TO E
$E                +id$  REDUCE TO E
$E+                id$  SHIFT->symbols
$E+id                $  SHIFT->id
$E+E                 $  REDUCE TO E
$E                   $  REDUCE TO E

id+(id*id)+id accepted.
```

Program Code

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

char code[20][20];

void main()
{
     printf("\nEnter the number of lines : ");
     int n;
     scanf("%d",&n);

     printf("\nEnter the codes : \n");

     for(int i=0;i<n;i++)
          scanf(" %s",code[i]);

     printf("\nAfter constant_propogation is applied :\n");

     for(int i=0;i<n;i++)
     {
          char temp[20];
          char c;
          strcpy(temp,"");
          int flag=1;
          for(int j=2;j<strlen(code[i]);j++)
          {
               c=code[i][0];
               if(code[i][j]>='0' && code[i][j]<='9')
               {
                    strncat(temp,&code[i][j],1);
               }
               else
               {
                    flag=0;
                    break;
               }
               //printf("\t%s",temp);
          }
          if(flag)
          {
               for(int j=i;j<n;j++)
               {
                    //printf("%d ",i);
                    for(int k=2;k<strlen(code[j]);k++)
                    {
                         if(code[j][k]==c)
                         {
                              char t[20];
                              strcpy(t,"");
                              for(int x=k+1;x<strlen(code[j]);x++)
                              {
                                   strncat(t,&code[j][x],1);
                              }

                              //printf("\n%s",t);
                              char r[20];
                              strcpy(r,"");
                              for(int x=0;x<k;x++)
```

```
                        {
                            strncat(r,&code[j][x],1);
                        }
                        //printf("%s",r);
                        strcat(r,temp);
                        strcat(r,t);
                        strcpy(code[j],r);
                        //printf("\n%s",code[j]);
                    }
                }
            }
        }
        else
            continue;

    }
    for(int i=0;i<n;i++)
        printf("\n%s",code[i]);
    printf("\n");
}
```

Output

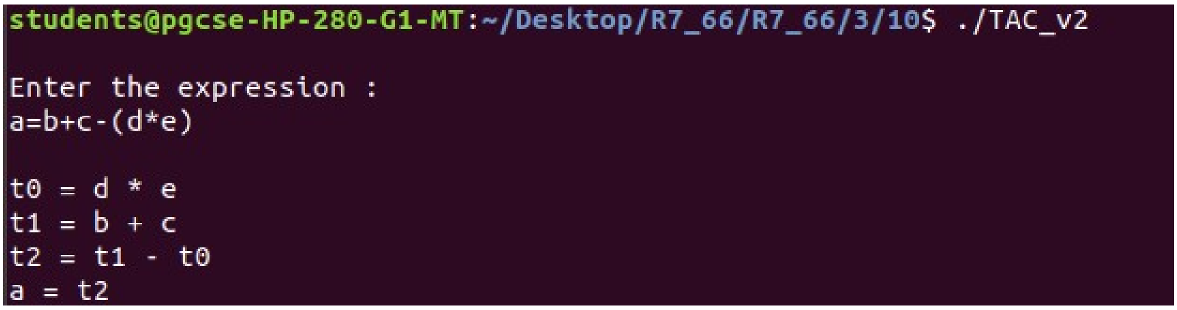


```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/3/9$ ./constant_propogation

Enter the number of lines : 5

Enter the codes :
a=5
b=3
c=a+b
d=2*a+b
e=d/a

After constant_propogation is applied :

a=5
b=3
c=5+3
d=2*5+3
e=d/5
```

Program Code

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

struct TAC
{
     char LHS;
     char op;
     char x;
     char y;
};

struct TAC tac[50];
char expr[50],ex[50];
int inc=1;

int k=0;

int priority(char x)
{
    if(x == '+' || x == '-')
        return 1*inc;
    if(x == '*' || x == '/')
        return 2*inc;
    return 0;
}


void convert()
{
  while(strlen(ex)>1)
  {
  int max=-1;
       for(int i=0;ex[i]!='\0';i++)
    {
        if(ex[i]=='(') inc++;
        else if(ex[i]==')') inc--;
        if(max!=-1 && priority(ex[max])<priority(ex[i]))
          max=i;
        else if(max==-1)
          max=i;

    }
          if(max!=-1)
          {
               tac[k].LHS='0'+k;
               tac[k].op=ex[max];
               tac[k].x=ex[max-1];
               tac[k].y=ex[max+1];

               ex[max-1]=tac[k].LHS;
               k++;
               for(int i=max;ex[i+2]!='\0';i++)
               {
                    ex[i]=ex[i+2];
               }
               ex[strlen(ex)-2]='\0';
        if(ex[max-2]=='(' && ex[max]==')')
        {
          ex[max-2]=ex[max-1];
          for(int i=max-1;i<strlen(ex)-2;i++)
            {
              ex[i]=ex[i+2];
            }
          ex[strlen(ex)-2]='\0';

        }
        }

     }
}

void main()
{
     printf("\nEnter the expression :\n");
     scanf(" %s",expr);
     int r = strlen(expr);

     strncat(ex,&expr[2],strlen(expr)-2);
     convert();

     for(int i=0;i<k;i++)
     {
          printf("\n");
          if(tac[i].LHS >='0' && tac[i].LHS <='9')
          {
               printf("t%c = ",tac[i].LHS);
          }
          else
          {
               printf("%c = ",tac[i].LHS);
          }
          if(tac[i].x >='0' && tac[i].x <='9')
          {
               printf("t%c ",tac[i].x);
          }
          else
          {
               printf("%c ",tac[i].x);
          }

          printf("%c ",tac[i].op);

          if(tac[i].y >='0' && tac[i].y <='9')
          {
               printf("t%c ",tac[i].y);
          }
          else
          {
               printf("%c ",tac[i].y);
          }
     }

     printf("\n%c = t%c\n",expr[0],tac[k-1].LHS);

}
```

Output

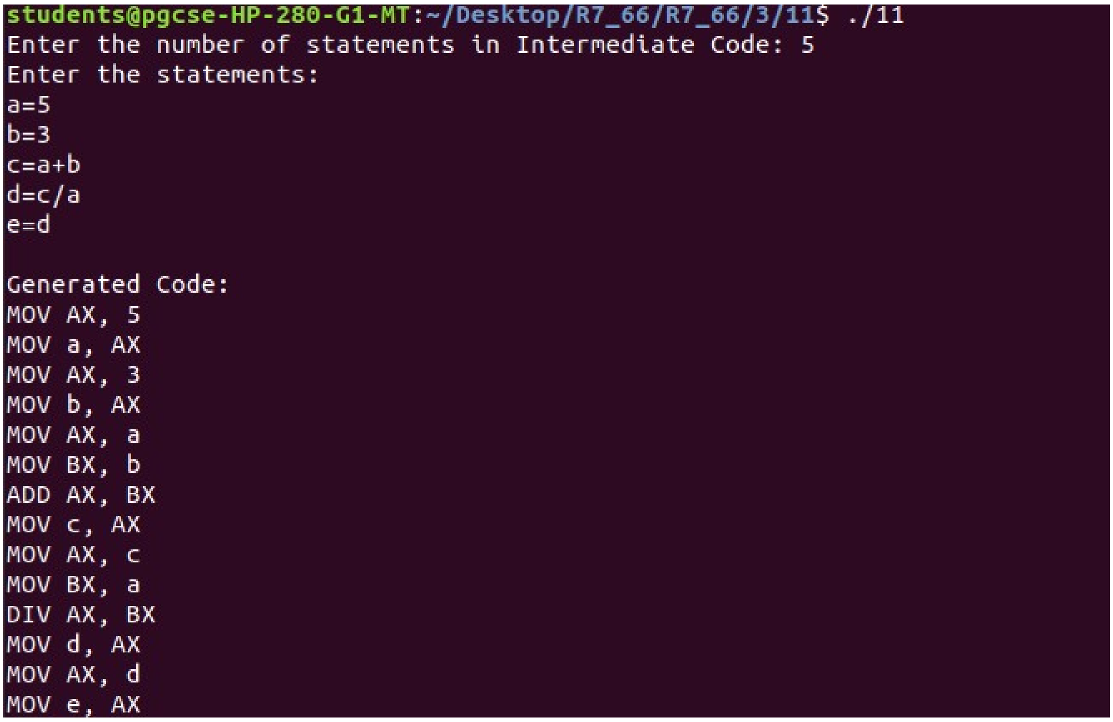


```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/3/10$ ./TAC_v2

Enter the expression :
a=b+c-(d*e)

t0 = d * e
t1 = b + c
t2 = t1 - t0
a = t2
```

Program Code

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

struct statement
{
    char lhs[2];
    char rhs[4];
};

void move(char a, char b)
{
     printf("MOV AX, %c\n", b);
     printf("MOV %c, AX\n", a);
}

void add(char a, char b, char c)
{
     printf("MOV AX, %c\n", b);
     printf("MOV BX, %c\n", c);
     printf("ADD AX, BX\n");
     printf("MOV %c, AX\n", a);
}

void sub(char a, char b, char c)
{
     printf("MOV AX, %c\n", b);
     printf("MOV BX, %c\n", c);
     printf("SUB AX, BX\n");
     printf("MOV %c, AX\n", a);
}

void mul(char a, char b, char c)
{
     printf("MOV AX, %c\n", b);
     printf("MOV BX, %c\n", c);
     printf("MUL AX, BX\n");
     printf("MOV %c, AX\n", a);
}

void div(char a, char b, char c)
{
     printf("MOV AX, %c\n", b);
     printf("MOV BX, %c\n", c);
     printf("DIV AX, BX\n");
     printf("MOV %c, AX\n", a);
}

void main()
{
     int n;

     printf("Enter the number of statements in Intermediate Code: 
");
     scanf("%d", &n);
     struct statement statements[n];

     printf("Enter the statements:\n");
     for(int i = 0; i < n; i++)
    {
        scanf(" %s, %s", statements[i].lhs, statements[i].rhs);
    }

    printf("\nGenerated Code:\n");
    for(int i = 0; i < n; i++)
    {
        if(strlen(statements[i].rhs) == 1)
        {
            move(statements[i].lhs[0], statements[i].rhs[0]);
        }
        else
        {
            switch(statements[i].rhs[1])
            {
            case '+':
                add(statements[i].lhs[0], statements[i].rhs[0],
statements[i].rhs[2]);
                break;
            case '-':
                sub(statements[i].lhs[0], statements[i].rhs[0],
statements[i].rhs[2]);
                break;
            case '*':
                mul(statements[i].lhs[0], statements[i].rhs[0],
statements[i].rhs[2]);
                break;
            case '/':
                div(statements[i].lhs[0], statements[i].rhs[0],
statements[i].rhs[2]);
                break;
            default:
                printf("Invalid statement!\n");
                return;
            }
        }
    }
}
```

Output



```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/3/11$ ./11
Enter the number of statements in Intermediate Code: 5
Enter the statements:
a=5
b=3
c=a+b
d=c/a
e=d

Generated Code:
MOV AX, 5
MOV a, AX
MOV AX, 3
MOV b, AX
MOV AX, a
MOV BX, b
ADD AX, BX
MOV c, AX
MOV AX, c
MOV BX, a
DIV AX, BX
MOV d, AX
MOV AX, d
MOV e, AX
```