**CODE**

#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("epinput.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:");

    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;

}

**INPUT FILE – EPINPUT.TXT**

q0 e q1

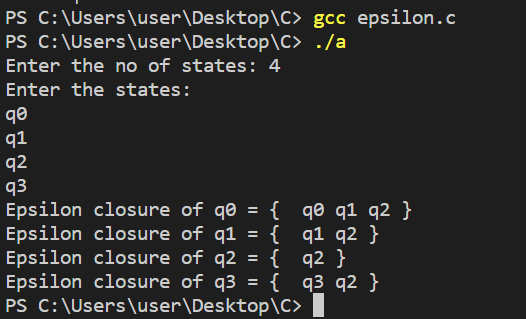
q1 e q2

q1 0 q2

q2 1 q3

q3 e q2

**OUTPUT**



**CODE**

#include<stdio.h>

#include<string.h>

char result[10][10],copy[3],states[20][20], alphabets[20][20];

int ind=0;

int notin(char state[3], char arraytrans[10][10], int limit){

    for(int i = 0;i<limit;i++){

        if(strcmp(arraytrans[i],state)==0)

        return 0;

    }

    return 1;

}

int main(){

    FILE \*INPUT1, \*INPUT2, \*INPUT3;

    INPUT1=fopen("epinput.txt", "r");

    INPUT2=fopen("epclosure.txt", "r");

    INPUT3=fopen("epclosure.txt ", "r");

    char state[3];

    int end1, end2,i=0,n,k=0, nalp;

    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]);

    }

    printf("Enter number of alphabets: ");

    scanf("%d", &nalp);

    printf("Enter the alphabets: ");

    for(k=0;k<nalp;k++){

        scanf("%s", alphabets[k]);

    }

    printf("The corresponding nfa without epsilon is:\n");

    printf("STATE\t\tINPUT\t\tSTATES\n");

    for(k=0;k<n;k++){

        char epclosure[4][10];

        char transition[10][10];

        int q=0;

        end1 = fscanf(INPUT2, "%s %s %s %s", epclosure[0], epclosure[1], epclosure[2], epclosure[3]);

        if(end1 == EOF)

        break;

        for(int i =0;i<nalp;i++){

            printf("%s\t\t%s\t\t", states[k], alphabets[i]);

            for(int j=0;j<n;j++){

                if(strcmp(epclosure[j], "-") != 0){

                    while(1){

                        end2 = fscanf(INPUT1, "%s %s %s", state1, input, state2);

                        if(end2 == EOF)

                        break;

                        if(strcmp(state1, epclosure[j]) == 0 && strcmp(input, alphabets[i])==0 && notin(state2, transition, q)!=0){

                            strcpy(transition[q], state2);

                            q++;

                        }

                    }

                    rewind(INPUT1);

                }

            }

            for(int j=0;j<q;j++){

                char fstates[10][10];

                while(1){

                    int isover = fscanf(INPUT3, "%s %s %s %s", fstates[0], fstates[1], fstates[2], fstates[3]);

                    if(isover == EOF)

                    break;

                    if(strcmp(transition[j], fstates[0]) == 0){

                        for(int a = 0;a<n;a++){

                            if(strcmp(fstates[a], "-")!=0 && notin(fstates[a], result, ind )!=0){

                                strcpy(result[ind], fstates[a]);

                                ind++;

                                printf("%s ", fstates[a]);

                            }

                        }

                    }

                }

                rewind(INPUT3);

            }

            if(ind == 0){

                printf("empty set");

            }

            printf("\n");

            for(int j=0;j<ind;j++){

                strcpy(result[j], " ");

            }

            ind = 0;

        }

    }

    return 0;

}

**INPUT FILE – EPINPUT.TXT**

q0 e q1

q1 e q2

q1 0 q2

q2 1 q3

q3 e q2

**INPUT FILE – EPCLOSURE.TXT**

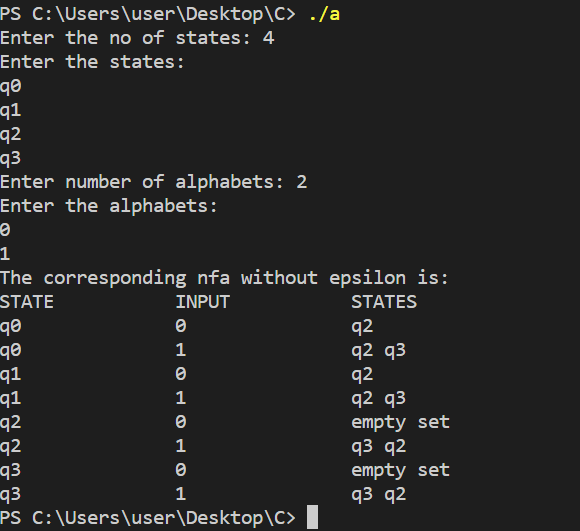
q0 q1 q2 -

q1 q2 - -

q2 - - -

q3 q2 - -

**OUTPUT**

****

**CODE**

#include<stdio.h>

#include<string.h>

int i=0, j=0, end;

struct dfastates{

    char state[10];

    int check;

};

struct dfastates st[10];

int notin(char s[], struct dfastates st[], int limit){

    for(int k = 0;k<limit;k++){

        if(strcmp(st[k].state,s)==0){

            return 0;

        }

    }

    return 1;

}

int checkdup(char state[], char states[10][10], int limit){

    for(int k=0;k<limit;k++){

        if(strcmp(state, states[k])==0)

        return 1;

    }

    return 0;

}

void main(){

    FILE\* f1;

    char state[10], input0[10], input1[10], delim[] = ",";

    f1 = fopen("nfasample.txt", "r");

    printf("The converted DFA is:\n");

    printf("STATE1\t\tinput\t\tSTATE2\n");

    end = fscanf(f1, "%s %s %s", state, input0, input1);

    if(notin(state, st, i)==1){

        strcpy(st[i].state, state);

        st[i].check = 1;

        j++;

        i++;

    }

    if(strcmp(input0, "-") != 0 && notin(input0, st, i) == 1){

        strcpy(st[i].state, input0);

        st[i].check = 0;

        i++;

    }

    if(strcmp(input1, "-") != 0 && notin(input1, st, i) == 1){

        strcpy(st[i].state, input1);

        st[i].check = 0;

        i++;

    }

    printf("%s\t\t%s\t\t%s\n", state, input0, input1);

    while(j<i){

        if(st[j].check == 0){

            printf("%s\t\t", st[j].state);

            char copy[10], result0[20], result1[20];

            int res0=0, res1=0;

            strcpy(copy, st[j].state);

            char\* ptr = strtok(copy, delim);

            while(ptr!=NULL){

                rewind(f1);

                end = fscanf(f1, "%s %s %s", state, input0, input1);

                while(end!=EOF){

                    if(strcmp(state, ptr)==0){

                        if(res0==0 && strcmp(input0, "-")!=0){

                            strcpy(result0, input0);

                            res0++;

                        }

                        else if(strcmp(input0, "-")!=0){

                            strcat(result0, ",");

                            strcat(result0, input0);

                        }

                        if(res1==0 && strcmp(input1, "-")!=0){

                            strcpy(result1, input1);

                            res1++;

                        }

                        else if(strcmp(input1, "-")!=0){

                            strcat(result1, ",");

                            strcat(result1, input1);

                        }

                    }

                    end = fscanf(f1, "%s %s %s", state, input0, input1);

                }

                ptr = strtok(NULL, delim);

            }

            st[j].check = 1;

            j++;

            char copy0[10][10], copy1[10][10], dupremove[10];

            //Remove the duplicates generated while concatenating

            char\* ptr0 = strtok(result0, delim);

            int count = 0;

            while(ptr0!=NULL){

                if(checkdup(ptr0, copy0, count)==0){

                    strcpy(copy0[count], ptr0);

                    count++;

                }

                ptr0 = strtok(NULL, delim);

            }

            strcpy(dupremove, copy0[0]);

            for(int k=1;k<count;k++){

                strcat(dupremove, ",");

                strcat(dupremove, copy0[k]);

            }

            printf("%s\t\t", dupremove);

            if(notin(dupremove, st, i)==1){

                strcpy(st[i].state, dupremove);

                st[i].check = 0;

                i++;

            }

            char\* ptr1 = strtok(result1, delim);

            count = 0;

            while(ptr1!=NULL){

                if(checkdup(ptr1, copy1, count)==0){

                    strcpy(copy1[count], ptr1);

                    count++;

                }

                ptr1 = strtok(NULL, delim);

            }

            strcpy(dupremove, copy1[0]);

            for(int k=1;k<count;k++){

                strcat(dupremove, ",");

                strcat(dupremove, copy1[k]);

            }

            printf("%s\n", dupremove);

            if(notin(dupremove, st, i)==1){

                strcpy(st[i].state, dupremove);

                st[i].check = 0;

                i++;

            }

        }

    }

}

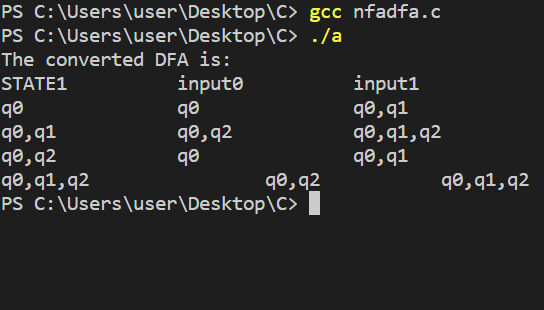
**INPUT FILE – NFASAMPLE.TXT**

q0 q0 q0,q1

q1 q2 q2

q2 - -

**OUTPUT**

****

**CODE**

#include <stdio.h>

#include <stdlib.h>

static int nostate,noalpha,s,notransition,nofinal,start,finalstate[20],r;

char alphabet[20];

int transition\_map[30][30], table[30][30], nonfinalstate[20], partition[20][20];

int findalpha(char a){

    int i;

    for(i=0;i<noalpha;i++)

        if(alphabet[i]==a)

            return i;

    return(-1);

}

int notin(int st, int states[], int limit){

    for(int k=0;k<limit;k++){

        if(st==states[k])

        return 0;

    }

    return 1;

}

int main(){

    int i,j,p[20],q[20],k, addst[20], addind=0;

    char a;

    for(i=0;i<30;i++){

        for(j=0;j<30;j++)

            transition\_map[i][j]=-1;

    }

    printf("Enter the number of alphabets: ");

    scanf("%d",&noalpha);

    getchar();

    printf("Enter the alphabets: \n");

    for(i=0;i<noalpha;i++)

    {

        alphabet[i]=getchar();

        getchar();

    }

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

    scanf("%d",&nostate);

    printf("Enter the start state: ");

    scanf("%d",&start);

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

    scanf("%d",&nofinal);

    printf("Enter the final states:\n");

    for(i=0;i<nofinal;i++)

        scanf("%d",&finalstate[i]);

    printf("Enter no of transition: ");

    scanf("%d",&notransition);

    printf("Enter Transition in the form: state alphabet next\_state\n");

    for(i=0;i<notransition;i++)

    {

        scanf("%d %c %d",&r,&a,&s);

        j=findalpha(a);

        if (j==-1){

            printf("\nerror\n"); exit(1);

        }

        transition\_map[r][j] = s;

    }

    //Initializing the upper trainagle as 0

    for(i=0;i<nostate;i++){

        for(j=0;j<i;j++){

            table[i][j]=0;

        }

    }

    //Finding the non-final states

    int f;

    k=0;

    for(i=0;i<nostate;i++){

        f=0;

        for(j=0;j<nofinal;j++){

            if(i==finalstate[j]){

                f=1;break;

            }

        }

        if(f==0){

            nonfinalstate[k++]=i;

        }

    }

    //Updating table values

    for(i=0;i<nofinal;i++){

        for(j=0;j<(nostate-nofinal);j++)

            if(nonfinalstate[j]>finalstate[i])

                table[nonfinalstate[j]][finalstate[i]]=1;

            else

                table[finalstate[i]][nonfinalstate[j]]=1;

    }

    //Updating the remaining table entries

    int change = 1;

    while(change==1){

        change=0;

        for(i=0;i<nostate;i++){

            for(j=0;j<i;j++){

                if(table[i][j]!=1){

                    for(k=0;k<noalpha;k++)

                        p[k]=transition\_map[i][k];

                    for(k=0;k<noalpha;k++)

                        q[k]=transition\_map[j][k];

                    for(k=0;k<noalpha;k++){

                        if(p[k]>q[k]){

                            if (table[p[k]][q[k]]==1){

                                change=1;

                                table[i][j]=1;

                                break;

                            }

                        }

                        else if(p[k]<q[k]){

                            if (table[q[k]][p[k]]==1){

                                change=1;

                                table[i][j]=1;

                                break;

                            }

                        }

                    }

                }

            }

        }

    }

    //Partition the states and display minimized states

    for(i=0;i<nostate;i++){

        k=0;

        partition[i][k++]=i;

        for(j=0;j<i;j++)

            if(table[i][j]==0){

                partition[i][k++]=j;

                addst[addind++]=j;

            }

        partition[i][k]=-1;

    }

    printf("\nThe final table");

    printf("\n----------------------------\n");

    for(int a=0;a<nostate;a++){

        for(int b=0;b<nostate;b++){

            printf("%d\t", table[a][b]);

        }

        printf("\n");

    }

    printf("\nStates in minimized DFA");

    printf("\n----------------------------\n");

    for(j=0;j<nostate;j++){

        if(notin(partition[j][0], addst, addind)==1){

            printf("{");

            int a = 0;

            while(partition[j][a]!=-1){

                printf("%d ", partition[j][a]);

                a++;

            }

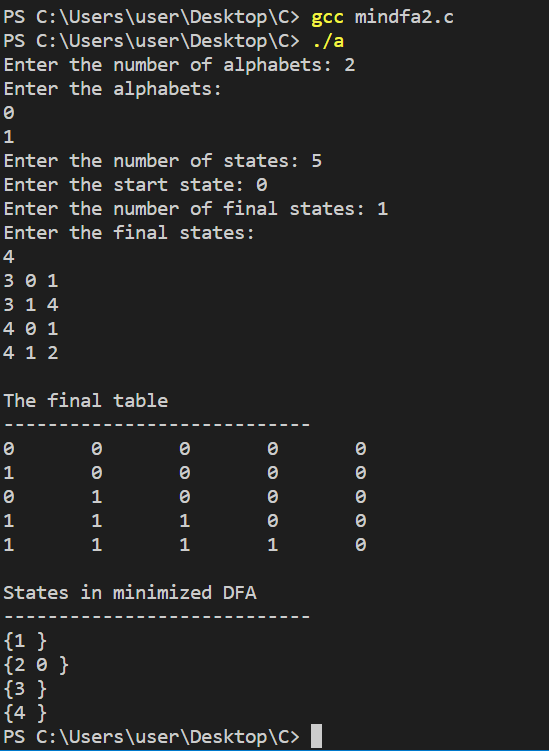
            printf("}\n");

        }

    }

}

**OUTPUT**



**CODE**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

int isKeyword(char buffer[])

{

    char keywords[32][10] = {"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"};

    for(int i = 0; i < 32; ++i)

    {

        if(strcmp(keywords[i], buffer) == 0)

        {

            return 1;

        }

    }

    return 0;

}

int main()

{

    int j = 0;

    char c, buffer[31], operators[] = "+-\*/%=<>!&|", separators[]=",;{}()";

    FILE \*fp;

    fp = fopen("inputlex.txt","r");

    if(fp == NULL)

    {

        printf("Error while opening the file\n");

        exit(0);

    }

    while((c = fgetc(fp)) != EOF)

    {

        if(c == '/')

        {

            char t = c;

            c = fgetc(fp);

            if(c == '/')

            {

                while(c != '\n')

                    c = fgetc(fp);

            }

            else

            {

                ungetc(c,fp);

                c = t;

            }

        }

        for(int i = 0; i < 12; ++i)

        {

            if(c == operators[i])

            {

                if(c == '+')

                {

                    c = fgetc(fp);

                    if(c == '+')

                        printf("++ is Increment operator\n");

                    else

                    {

                        ungetc(c,fp);

                        printf("+ is Addition operator\n");

                    }

                }

                else if(c == '-')

                {

                    c = fgetc(fp);

                    if(c == '-')

                        printf("-- is Decrement operator\n");

                    else

                    {

                        ungetc(c,fp);

                        printf("- is Subtraction operator\n");

                    }

                }

                else if(c == '\*')

                    printf("%c is Mutiplication operator\n", c);

                else if(c == '/')

                    printf("%c is Division operator\n", c);

                else if(c == '%')

                    printf("%c is Modulus operator\n", c);

                else if(c == '=')

                {

                    c = fgetc(fp);

                    if(c == '=')

                        printf("== is Relational operator\n");

                    else

                    {

                        ungetc(c,fp);

                        printf("= is Assignment operator\n", c);

                    }

                }

                else if(c == '!')

                {

                    c = fgetc(fp);

                    if(c == '=')

                        printf("!= is Relational operator\n");

                    else

                        printf("! is logical operator");

                }

                else if((c == '<') || (c == '>'))

                {

                    char t = c;

                    c = fgetc(fp);

                    if(c == '=')

                        printf("%c%c is Relational operator\n", t,c);

                    else

                    {

                        ungetc(c,fp);

                        printf("%c is Relational operator\n", t);

                    }

                }

                else if(c == '&' || c== '|')

                {

                    char t = c;

                    c = fgetc(fp);

                    if(c == t)

                        printf("%c%c is Logical operator\n", t,c);

                    else{

                        ungetc(c,fp);

                        printf("%c is a bitwise operator\n", t);

                    }

                }

            }

        }

        for(int i = 0; i < 6; ++i)

        {

            if(c == separators[i])

            {

                if(c == ';')

                    printf("%c is Delimiter\n", c);

                else

                    printf("%c is Separator\n", c);

            }

        }

        if(isalnum(c))

            buffer[j++] = c;

        else if((c == ' ' || c =='\t' || c == '\n') && (j != 0))

        {

            buffer[j] = '\0';

            j = 0;

            if(isKeyword(buffer) == 1)

                printf("%s is keyword\n", buffer);

            else

                printf("%s is identifier\n", buffer);

        }

    }

    fclose(fp);

    return 0;

}

**INPUT FILE – INPUTLEX.TXT**

void main ()

{

// Sample Program

int num1 , num2 , num3 ;

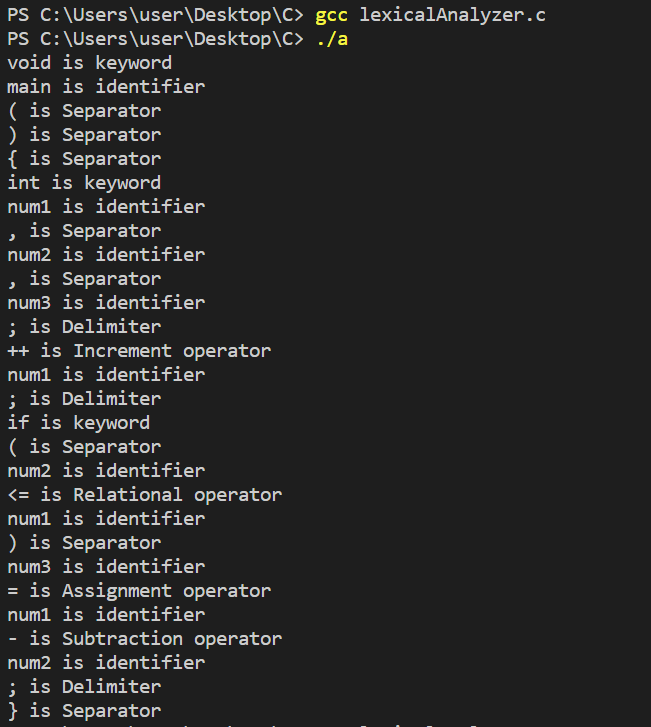
++num1 ;

if ( num2 <= num1 )

num3 = num1 - num2 ;

}

**OUTPUT**



**CODE**

%{

#include<stdio.h>

#include<stdlib.h>

int flag = 0;

int yyerror();

%}

%%

. {

flag = 1;

yyerror();

}

(1(01\*0)\*1|0)\* {

printf("The given binary number is divisible by 3- %s\n", yytext);

}

%%

int yywrap(){

return 1;

}

int main(){

yylex();

return 0;

}

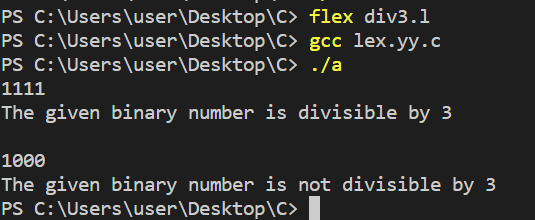
int yyerror(){

printf("The given binary number is not divisible by 3\n", yytext);

exit(0);

}

**OUTPUT**



**CODE**

**Lex file**

%{

/\* Definition section \*/

#include "calc.tab.h"

extern int yylval;

%}

/\* Rule Section \*/

%%

[0-9]+ {

yylval=atoi(yytext);

return NUMBER;

}

[\t] ;

[\n] return 0;

[+,\*,/,%,-] return yytext[0];

. {

yyerror();

}

%%

int yywrap()

{

return 1;

}

**Yacc file**

%{

/\* Definition section \*/

#include<stdio.h>

int flag=0;

int yylex();

int yyerror(const char \*s);

%}

%token NUMBER

%left '+' '-'

%left '\*' '/' '%'

%left '(' ')'

/\* Rule Section \*/

%%

ArithmeticExpression: E{

if (flag==0)

printf("\nResult=%d\n", $$);

return 0;

};

E:E'+'E {$$=$1+$3;}

|E'-'E {$$=$1-$3;}

|E'\*'E {$$=$1\*$3;}

|E'/'E {$$=$1/$3;}

|E'%'E {$$=$1%$3;}

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

| NUMBER {$$=$1;}

;

%%

//driver code

void main()

{

printf("\nEnter Any Arithmetic Expression: ");

yyparse();

if(flag==0)

printf("\nEntered arithmetic expression is Valid\n\n");

}

int yyerror(const char \*s)

{

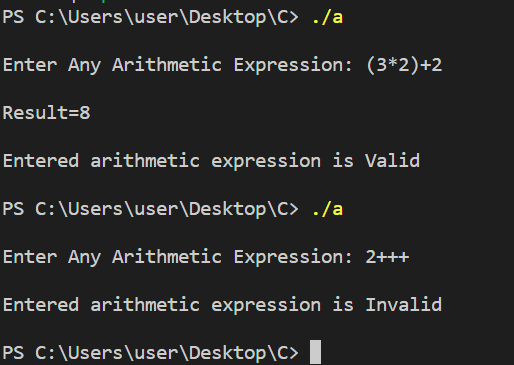
printf("\nEntered arithmetic expression is Invalid\n\n");

flag = 1;

return 0;

}

**OUTPUT**



**CODE**

//This is the recursive descent parser for the statement:

//if ( c ) { if ( c ) {...if ( c ) { stmt ; } else { stmt ; }...} else { stmt ; } } else { stmt; }

//The corresponding grammar is:

//S' --> S | SS'

//S --> B | B else { S } | A

//B --> if ( c ) { S }

//A --> stmt;

//Enter $ to indicate end of input

#include<stdio.h>

#include<string.h>

char\* S(char \*\_token, const char s[2]);

char\* \_S(char \*\_token, const char s[2]);

//function to move pointer

char\* moveip(char \*\_token, const char s[2]){

    if(\_token != "$"){

        return strtok(NULL, s);

    }

    else{

        return NULL;

    }

}

//function for symbol A

//A --> stmt ;

char\* A(char \*\_token, const char s[2]){

    int value;

    if(\_token != NULL && strcmp(\_token, "stmt")==0){

        \_token = moveip(\_token,s);

        if(\_token != NULL && strcmp(\_token, ";")==0){

            \_token = moveip(\_token,s);

            return \_token;

        }

        else{

            return NULL;

        }

    }

    else{

        return NULL;

    }

}

//function for symbol B

//B --> if ( c ) { S }

char\* B(char \*\_token, const char s[2]){

    int value;

    if(\_token != NULL && strcmp(\_token, "if") == 0){

        \_token = moveip(\_token, s);

        if(\_token != NULL && strcmp(\_token, "(") == 0){

            \_token = moveip(\_token, s);

            if(\_token != NULL && strcmp(\_token, "c") == 0){

                \_token = moveip(\_token, s);

                if(\_token != NULL && strcmp(\_token, ")") == 0){

                    \_token = moveip(\_token, s);

                    if(\_token != NULL && strcmp(\_token, "{") == 0){

                        \_token = moveip(\_token, s);

                        \_token = S(\_token, s);

                        if(\_token != NULL && strcmp(\_token,"}") == 0){

                            \_token = moveip(\_token, s);

                            return \_token;

                        }

                        else{

                            return NULL;

                        }

                    }

                    else{

                        return NULL;

                    }

                }

                else{

                    return NULL;

                }

            }

            else{

                return NULL;

            }

        }

        else{

            return NULL;

        }

    }

    else{

        return NULL;

    }

}

//function for start symbol S

//S --> B | B else { S } | A

char\* S(char \*\_token, const char s[2]){

    if(strcmp(\_token, "$")==0){

        return \_token;

    }

    char\* temptok;

    temptok = B(\_token, s);

    if(temptok != NULL){

        if(strcmp(temptok, "else")==0){

            temptok = moveip(temptok, s);

            if(temptok != NULL && strcmp(temptok, "{")==0){

                temptok = moveip(temptok, s);

                temptok = S(temptok, s);

                if(temptok != NULL && strcmp(temptok, "}")==0){

                    temptok = moveip(temptok, s);

                    \_token = temptok;

                    return \_token;

                }

                else{

                    return NULL;

                }

            }

            else{

                return NULL;

            }

        }

        else{

            \_token = temptok;

            return \_token;

        }

    }

    temptok = A(\_token, s);

    if(temptok != NULL){

        if(strcmp(temptok, "}")==0 || strcmp(temptok, "$")==0){

            \_token = temptok;

            return \_token;

        }

        return \_token;

    }

}

//function for starting symbol S'

//S' --> S | SS'

char\* \_S(char \*\_token, const char s[2]){

    if(strcmp(\_token, "$")==0){

        return \_token;

    }

    char\* temptok;

    temptok = S(\_token, s);

    if(temptok != NULL){

        if(strcmp(temptok, "$")==0){

            return temptok;

        }

        temptok = \_S(temptok, s);

    }

}

//main function

void main(){

    char command[100];

    int invalid = 0;

    printf("Enter command:\n");

    scanf("%[^\n]", command);

    const char s[2] = " ";

    char \*token;

    /\* get the first token \*/

    token = strtok(command, s);

    //call the function defined for start symbol

    char\* final = \_S(token, s);

    if(final != NULL && strcmp(final, "$")==0){

        printf("Valid command\n");

    }

    else{

        printf("Invalid command\n");

    }

}

**OUTPUT**

