**1. Program to construct a DFA which accepts the language L = {aN | N ≥ 1}**

**Source Code**

#include <bits/stdc++.h>

using namespace std;

void isAcceptedDFA(string s, int N)

{

int count = 0;

// Iterate over the range [0, N]

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

// Count and check every

// element for 'a'

if (s[i] == 'a')

count++;

}

if (count == N && count != 0) {

cout << "Accepted";

}

// If not matches

else {

cout << "Not Accepted";

}

}

int main()

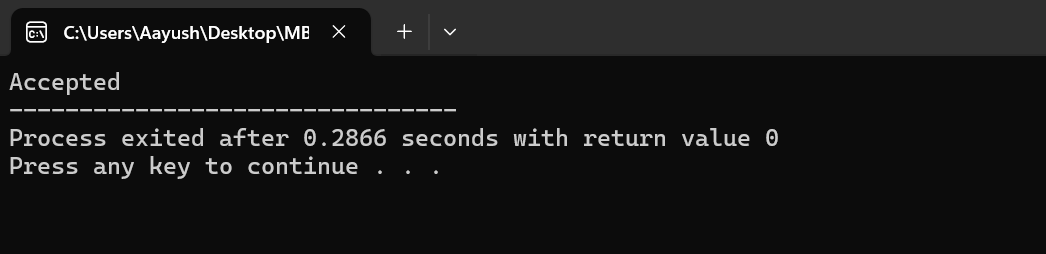
{

string S = "aaaaa";

isAcceptedDFA(S, S.size());

return 0;

}

****

**2. Program to construct a DFA which accept the language L = {anbm | n mod 2=0, m≥1}**

**Source Code**

// C program to implement DFS that accepts

// L = { anbm; (n)mod 2=0, m>=1 }

#include <stdio.h>

#include <string.h>

int dfa = 0;

void start(char c) {

if (c == 'a') {

dfa = 1;

} else if (c == 'b') {

dfa = 3;

} else {

dfa = -1;

}

}

void state1(char c) {

if (c == 'a') {

dfa = 2;

} else if (c == 'b') {

dfa = 4;

} else {

dfa = -1;

}

}

void state2(char c) {

if (c == 'b') {

dfa = 3;

} else if (c == 'a') {

dfa = 1;

} else {

dfa = -1;

}

}

void state3(char c) {

if (c == 'b') {

dfa = 3;

} else if (c == 'a') {

dfa = 4;

} else {

dfa = -1;

}

}

void state4(char c) {

dfa = -1;

}

int isAccepted(char str[]) {

int i, len = strlen(str);

for (i = 0; i < len; i++) {

if (dfa == 0)

start(str[i]);

else if (dfa == 1)

state1(str[i]);

else if (dfa == 2)

state2(str[i]);

else if (dfa == 3)

state3(str[i]);

else if (dfa == 4)

state4(str[i]);

else

return 0;

}

if (dfa == 3)

return 1;

else

return 0;

}

int main() {

char str[] = "aaaaaabbbb";

if (isAccepted(str))

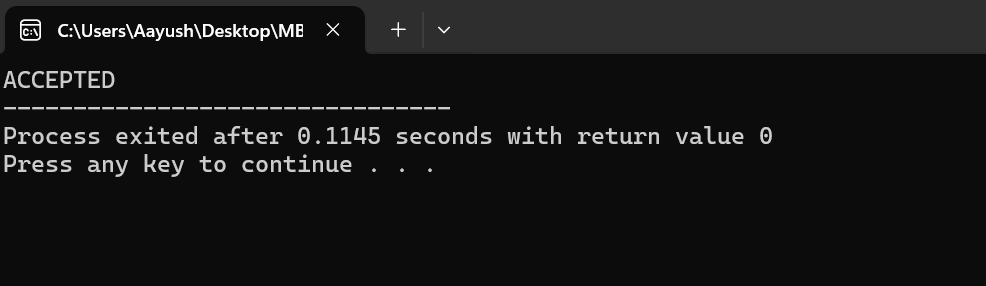
printf("ACCEPTED");

else

printf("NOT ACCEPTED");

return 0;

}



**3. Design deterministic finite automata (DFA) with ∑ = {0, 1} that accepts the languages ending with “01” over the characters {0, 1}.**

#include<stdio.h>

#include<conio.h>

#define max 100

main() {

char str[max],f='a';

int i;

printf("enter the string to be checked: ");

scanf("%s",str);

for(i=0;str[i]!='\0';i++) {

switch(f) {

case 'a': if(str[i]=='0') f='b';

else if(str[i]=='1') f='a';

break;

case 'b': if(str[i]=='0') f='b';

else if(str[i]=='1') f='c';

break;

case 'c': if(str[i]=='0') f='b';

else if(str[i]=='1') f='a';

break;

}

}

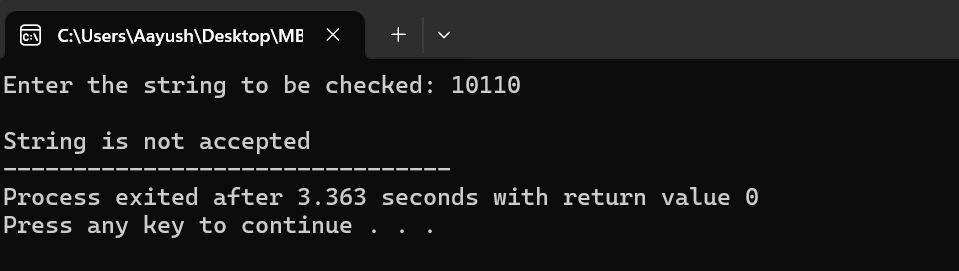
if(f=='c')

printf("\nString is accepted", f);

else printf("\nString is not accepted", f);

return 0;

}



**4.** **Construct deterministic finite automata (DFA) for the language L = { w : w has odd number of 0’s and w has odd number of 1’s},over the alphabet Σ = {0, 1}.**

#include <stdio.h>

int EE=0, OE=1, OO=2, EO=3;

int state = 0;

char input;

int main(void) {

printf("Enter a string of 0s and 1s: ");

while (1) {

scanf("%c", &input);

if (input == '\n')

break;

if ( (input != '0') && (input != '1') ) {

break;

}

if (state==0) {

}

else if(state==1) {

state = (input == '0') ? EE : OO;

}

else if (state==2) {

state = (input == '0') ? EO : OE;

} else {

state = (input == '0') ? OO : EE;

}

};

if (input == '\n') {

if (state == OO)

printf("Input accepted\n");

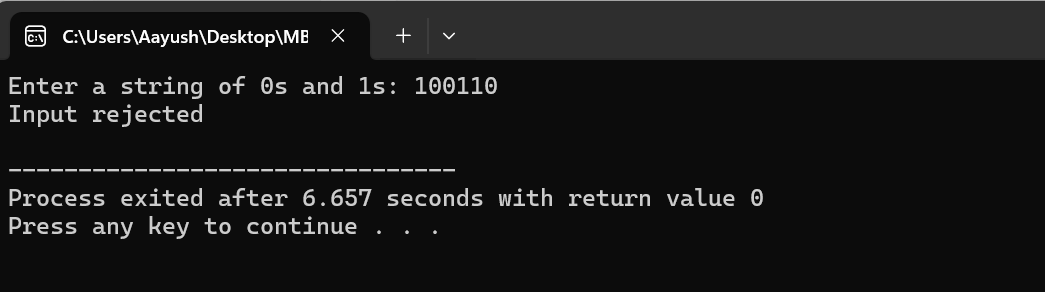
else

printf("Input rejected\n");

}

return 0;

}



**5. Design a DFA machine accepting odd numbers of 0’s or even numbers of 1’s**

**Source Code**

#include <stdio.h>

#include <string.h>

int transition(int current, char input) {

if (current == 0) {

if (input == '0') return 1;

else return 3;

} else if (current == 1) {

if (input == '0') return 0;

else return 3;

} else if (current == 2) {

if (input == '0') return 3;

else return 2;

} else {

if (input == '0') return 1;

else return 2;

}

}

int isAccepted(char str[]) {

int current = 0;

int i;

for (i = 0; str[i] != '\0'; i++) {

current = transition(current, str[i]);

}

return (current == 1 || current == 2);

}

int main() {

char str[] = "1010101";

if (isAccepted(str)) {

printf("String accepted\n");

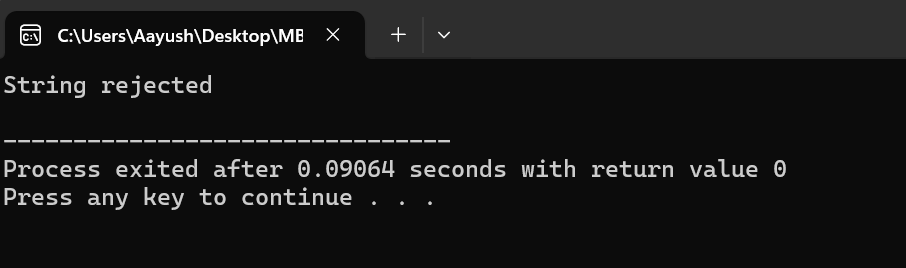
} else {

printf("String rejected\n");

}

return 0;

}



**6. WAP to simulate Nondeterministic Finite Automata (NFA)**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<stdbool.h>

#include<math.h>

int row = 0;

struct node {

int data;

struct node\* next;

char edgetype;

};

struct node\* push(struct node\* first, char edgetype, int data) {

struct node\* new\_node = (struct node\*)malloc(sizeof(struct node));

new\_node->edgetype = edgetype;

new\_node->data = data;

new\_node->next = NULL;

if (first == NULL) {

first = new\_node;

return new\_node;

}

first->next = push(first->next, edgetype, data);

return first;

}

int nfa(struct node\*\* graph, int current, char\* input, int\* accept, int start) {

if (start == (int)strlen(input))

return accept[current];

struct node\* temp = graph[current];

while (temp != NULL) {

if (input[start] == temp->edgetype) {

if (nfa(graph, temp->data, input, accept, start + 1 == 1))

return 1;

}

temp = temp->next;

}

return 0;

}

void generate(char\*\* arr, int size, char\* a) {

if (size == 0) {

strcpy(arr[row], a);

row++;

return;

}

char b0[20] = {'\0'};

char b1[20] = {'\0'};

b0[0] = '0';

b1[0] = '1';

generate((char\*\*)arr, size - 1, strcat(b0, a));

generate((char\*\*)arr, size - 1, strcat(b1, a));

return;

}

int main() {

int n;

int i, j;

scanf("%d", &n);

struct node\* graph[n + 1];

for (i = 0; i < n + 1; i++)

graph[i] = NULL;

int accept[n + 1]; // Array to store state of vertex

for (i = 0; i < n; i++) {

// Index of vertex, Acceptance state, Number of edges

int index, acc, number\_nodes;

scanf("%d%d%d", &index, &acc, &number\_nodes);

accept[index] = acc; // Store acceptance

for (j = 0; j < number\_nodes; j++) { // Add all edges

int node\_add;

int edge;

scanf("%d%d", &edge, &node\_add);

graph[index] = push(graph[index], '0' + edge, node\_add);

}

}

int size = 1;

int count = 0;

if (accept[1] == 1) {

printf("e\n");

count++;

}

while (count < 11) {

char\*\* arr;

int power = pow(2, size);

arr = (char\*\*)malloc(power \* sizeof(char\*));

for (i = 0; i < power; i++)

arr[i] = (char\*)malloc(size \* sizeof(char));

char a[20] = {'\0'};

generate((char\*\*)arr, size, a);

for (i = 0; i < power; i++) {

char input[20] = {'\0'};

for (j = 0; j < size; j++) {

char foo[2];

foo[0] = arr[i][size - 1 - j];

foo[1] = '\0';

strcat(input, foo);

}

int result = nfa(graph, 1, input, accept, 0); // Store result of nfa

if (result == 1) {

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

count++;

}

if (count == 10)

return 0;

}

size++;

row = 0;

}

return 0;

}

7. Program to implement NFA from Regular Expression

#include<stdio.h>

#include<conio.h>

int fa[10][10][10], states[2][10], curr, row = 0, col = 0, sr = 0, sc = 0, th = 0, in;

char str[100];

int nfa(char \*string, int state) {

int i, j;

for (i = 0; i <= row; i++) {

if (\*string) {

curr = fa[state][\*string - 97][i];

if (curr == -1)

break;

if (nfa(string + 1, curr))

return 1;

} else {

if (states[1][i] == -1)

break;

if (state == states[1][i])

return 1;

}

}

return 0;

}

int main() {

FILE \*fp;

int i, j, k, flag = 0;

char c;

clrscr();

fp = fopen("Nfa\_ip.txt", "r");

if (fp == NULL) {

printf("Error opening file\n");

return 1;

}

for (i = 0; i < 2; i++)

for (j = 0; j < 10; j++)

states[i][j] = -1;

for (i = 0; i < 10; i++)

for (j = 0; j < 10; j++)

for (k = 0; k < 10; k++)

fa[i][j][k] = -1;

while (fscanf(fp, "%d", &in) != EOF) {

fscanf(fp, " %c", &c); // Added a space before %c to consume leading whitespace

if (flag) {

states[sr][sc++] = in;

if (c == '\n') {

sr++;

sc = 0;

}

} else if (c == '#') {

flag = 1;

fa[row][col][th] = in;

printf("\nfa[%d][%d][%d]=%d", row, col, th, fa[row][col][th]);

} else if (!flag) {

fa[row][col][th] = in;

printf("\nfa[%d][%d][%d]=%d", row, col, th, fa[row][col][th]);

if (c == ',') {

th++;

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

col = 0;

row++;

th = 0;

} else if (c != ',') {

col++;

th = 0;

}

}

}

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

scanf("%s", str);

if (nfa(str, states[0][0]))

printf("\nString Is Accepted\n");

else

printf("\nString Not Accepted\n");

getch();

return 0;

}

**8. A simple and basic program in C to convert NFA to DFA (does not handle null moves)**

#include<stdio.h>

#include<string.h>

#include<math.h>

int ninputs;

int dfa[100][2][100] = {0};

int state[10000] = {0};

char ch[10], str[1000];

int go[10000][2] = {0};

int arr[10000] = {0};

int main() {

int st, fin, in;

int f[10];

int i,j=3,s=0,final=0,flag=0,curr1,curr2,k,l;

int c;

printf("\nFollow the one based indexing\n");

printf("\nEnter the number of states::");

scanf("%d",&st);

printf("\nGive state numbers from 0 to %d",st-1);

for(i=0;i<st;i++)

state[(int)(pow(2,i))] = 1;

printf("\nEnter number of final states\t");

scanf("%d",&fin);

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

for(i=0;i<fin;i++) {

scanf("%d",&f[i]);

}

int p,q,r,rel;

printf("\nEnter the number of rules according to NFA::");

scanf("%d",&rel);

printf("\n\nDefine transition rule as \"initial state input symbol final state\"\n");

for(i=0; i<rel; i++) {

scanf("%d%d%d",&p,&q,&r);

if (q==0)

dfa[p][0][r] = 1;

else

dfa[p][1][r] = 1;

}

printf("\nEnter initial state::");

scanf("%d",&in);

in = pow(2,in);

i=0;

printf("\nSolving according to DFA");

int x=0;

for(i=0;i<st;i++) {

for(j=0;j<2;j++) {

int stf=0;

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

if(dfa[i][j][k]==1)

stf = stf + pow(2,k);

}

go[(int)(pow(2,i))][j] = stf;

printf("%d-%d-->%d\n",(int)(pow(2,i)),j,stf);

if(state[stf]==0)

arr[x++] = stf;

state[stf] = 1;

}

}

// for new states

for(i=0;i<x;i++) {

printf("for %d ---- ",arr[x]);

for(j=0;j<2;j++) {

int new=0;

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

if(arr[i] & (1<<k)) {

int h = pow(2,k);

if(new==0)

new = go[h][j];

new = new | (go[h][j]);

}

}

if(state[new]==0) {

arr[x++] = new;

state[new] = 1;

}

}

}

printf("\nThe total number of distinct states are::\n");

printf("STATE 0 1\n");

for(i=0;i<10000;i++) {

if(state[i]==1) {

int y=0;

if(i==0)

printf("q0 ");

else

for(j=0;j<st;j++) {

int x = 1<<j;

if(x&i) {

printf("q%d ",j);

y = y+pow(2,j);

}

}

printf(" %d %d",go[y][0],go[y][1]);

printf("\n");

}

}

j=3;

while(j--) {

printf("\nEnter string");

scanf("%s",str);

l = strlen(str);

curr1 = in;

flag = 0;

printf("\nString takes the following path-->\n");

printf("%d-",curr1);

for(i=0;i<l;i++) {

curr1 = go[curr1][str[i]-'0'];

printf("%d-",curr1);

}

printf("\nFinal state - %d\n",curr1);

for(i=0;i<fin;i++) {

if(curr1 & (1<<f[i])) {

flag = 1;

break;

}

}

if(flag)

printf("\nString Accepted");

else

printf("\nString Rejected");

}

return 0;

}

