**PRACTICAL NO.: 01**

**Title:**

Implementation and Analysis of DFA for string recognition

**Objective:**

To Implement and design the finite state machine DFA for recognizing strings over alphabets {0,1}

**Theory:**

DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time. In DFA, there is only one path for specific input from the current state to the next state. DFA does not accept the null move, i.e., the DFA cannot change state without any input character. DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

The formal definition of DFA:

A DFA can be represented by a 5-tuple (Q, ∑, δ, q0, F) where −

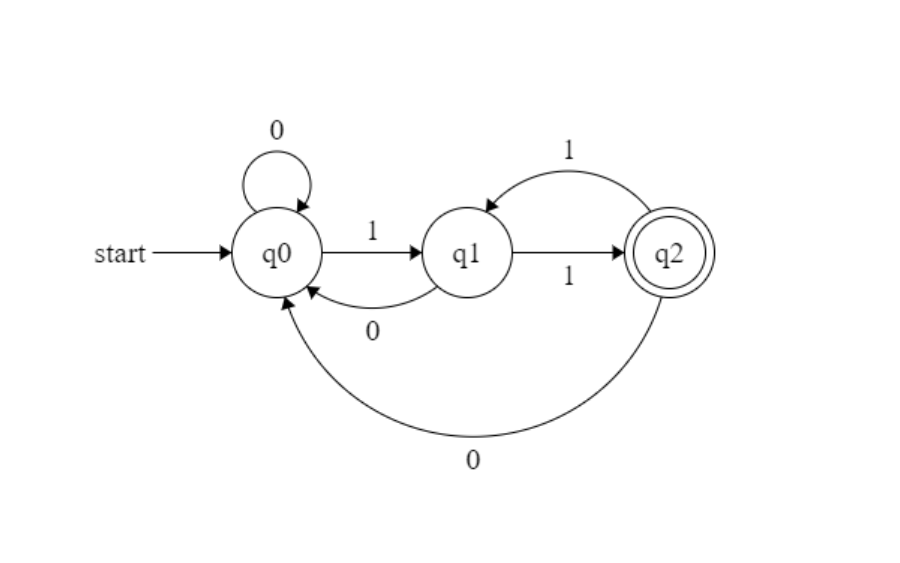
* Q is a finite set of states.
* ∑ is a finite set of symbols called the alphabet.
* δ is the transition function where δ: Q × ∑ → Q
* q0 is the initial state from where any input is processed (q0 ∈ Q).
* F is a set of final state/states of Q (F ⊆ Q).

A DFA that accept a language that end with string “11” is defined as:

* Q = {q0, q1, q2},
* ∑ = {0, 1},
* q0 = {q0},
* F = {q2}, and

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next State for Input 0** | **Next State for Input 1** |
| q0 | q0 | q1 |
| q1 | q0 | q2 |
| q2 | q0 | q1 |

Transition function δ as shown by the following table −

Its transition diagram would be as follows −

**Source Code:**

#include <iostream>

#include <string>

using namespace std;

const char STATE\_Q0 = 0;

const char STATE\_Q1 = 1;

const char STATE\_Q2 = 2;

int main()

{

string str; // string to be checked

char state = STATE\_Q0; // initial state (q0)

cout << "Enter the string: ";

cin >> str;

for (int i = 0; i < str.length(); i++)

{

if (str[i] != '0' && str[i] != '1'){

cout << "String not accepted.\nPlease enter a string over {0,1}" << endl;

return 0;

}

if (state == STATE\_Q0 && str[i] == '0')

state = STATE\_Q0;

else if (state == STATE\_Q0 && str[i] == '1')

state = STATE\_Q1;

else if (state == STATE\_Q1 && str[i] == '0')

state = STATE\_Q0;

else if (state == STATE\_Q1 && str[i] == '1')

state = STATE\_Q2;

else if (state == STATE\_Q2 && str[i] == '0')

state = STATE\_Q0;

else if (state == STATE\_Q2 && str[i] == '1')

state = STATE\_Q1;

}

if (state == STATE\_Q2)

cout << "String accepted";

else

cout << "String not accepted";

return 0;

}

**Output:**

Enter the string: 101011

String accepted

Enter the string: 100100

String not accepted

**PRACTICAL NO.: 02**

**Title:**

Implementation and Analysis of NFA for string recognition

**Objective:**

To Implement and design the finite state machine NFA for recognizing strings over alphabets {0,1}

**Theory:**

NFA refers to Nondeterministic Finite Automaton. A Finite Automata (FA) is said to be non-deterministic if there is more than one possible transition from one state on the same input symbol.

The formal definition of NFA:

A NFA can be represented by a 5-tuple (Q, ∑, δ, q0, F) where −

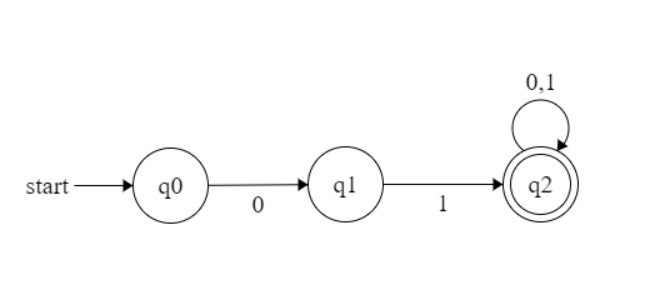
* Q is a finite set of states.
* ∑ is a finite set of symbols called the alphabet.
* δ is the transition function where δ: Q × ∑ ∪ {ε}→2|Q|
* q0 is the initial state from where any input is processed (q0 ∈ Q).
* F is a set of final state/states of Q (F ⊆ Q).

A NFA that accept a language that start with string “01” is defined as:

* Q = {q0, q1, q2},
* ∑ = {0, 1},
* q0 = {q0},
* F = {q2}, and

Transition function δ as shown by the following table −

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next State for Input 0** | **Next State for Input 1** |
| q0 | q1 | - |
| q1 | - | q1 |
| q2 | q2 | q2 |

Its transition diagram would be as follows −

**Source Code:**

#include<stdio.h>

#include<conio.h>

void q0(char);

void q1(char);

void q2(char);

int current=0;

void q0 (char c)

{

if(c=='0') current=1;

else current=-1;

}

void q1 (char c)

{

if(c=='1') current=2;

else current=-1;

}void q2 (char c)

{

if(c=='0') current=2;

else current=-1;

}

int main(){

char inputstr[20], c;

int i=0;

printf("\n Enter an input string"); gets(inputstr);

c=inputstr[i];

while(c!='\0'){

switch(current){

case 0: q0(c);

break;

case 1: q1(c);

break;

case 2: q2(c);

break;

}

c=inputstr[++i];

}

if(current==2){

printf("\nString accepted");

} else {printf("\n String not accepted");

}

getch();

}

**Output:**

Enter the string: 01010

String accepted

Enter the string: 10111

String not accepted

**PRACTICAL NO.: 03**

**Title:**

Identifiers and Keywords recognition Program

**Objective:**

To identify and validate the C identifiers and keywords

**Theory:**

**C identifiers:** These are the names of variables, functions, arrays, structures and pointers etc. The first character of C identifiers must be letter or underscore and remaining characters might be letters, digits or underscore.

**Keywords:** These are the reserved words having predefined meaning in the language. There are 32 keywords in C. They cannot be used as identifiers.

**Source Code**

#include<stdio.h>

#include<string.h>

char keyword[32][10] = {"auto", "double", "int", "struct", "break", "else",

"long", "switch", "case", "enum", "register", "typedef",

"char", "extern", "return", "union", "const", "float",

"short", "unsigned", "continue", "for", "signed", "void",

"default", "goto", "sizeof", "volatile", "do", "if",

"static", "while"};

enum states {q0, qf, qd};

enum states delta(enum states, char);

int iskeyword(char []);

int main(){

enum states curr\_state = q0;

char string[20], ch;

int i = 0;

printf("\n Enter a string \t");

gets(string);

ch = string[i];

if(iskeyword(string))

printf("\n The string %s is keyword.", string);

else {

while(ch != '\0') {

curr\_state = delta(curr\_state, ch);

ch = string[++i];

}

if(curr\_state == qf)

printf("\n The string %s is valid identifier.", string);

else

printf("\n The string %s is neither keyword nor valid identifier.", string);

}

return 0;

}

enum states delta(enum states s, char ch) {

enum states curr\_state;

switch(s) {

case q0:

if(ch >= 'A' && ch <= 'Z' || ch >= 'a' && ch <= 'z' || ch == '\_')

curr\_state = qf;

else

curr\_state = qd;

break;

case qf:

if(ch >= 'A' && ch <= 'Z' || ch >= 'a' && ch <= 'z' || ch == '\_' || ch >= '0' && ch <= '9')

curr\_state = qf;

else

curr\_state = qd;

break;

case qd:

curr\_state = qd;

break;

}

return curr\_state;

}

int iskeyword(char str[]) {

for(int i = 0; i < 32; i++) {

if(strcmp(str, keyword[i]) == 0)

return 1;

}

return 0;

}

**Output:**

Enter a string num

The string num is valid identifier.

Enter a string int

The string int is keyword

Enter a string 1abfc

The string 1abfc is neither keyword nor valid identifier.

**PRACTICAL NO.: 04**

**Title:**

Implementation and Analysis of PDA for string recognition

**Objective:**

To Implement and design the PDA for recognizing strings over alphabets {0,1}

**Theory:**

The context free languages have a type of automation that defines them. This automation is called “pushdown automation” which can be thought as a ε-NFA with the addition of stack. The presence of a stack means that, the pushdown automata can remember infinity amount of information. However, the pushdown automation can only access the information on its stack in a Last-in-first-out way.

A PDA is defined by seven tuples (Q, ∑, Γ, δ, q0, Z0, F) where-

* **Q** is the finite number of states
* **∑** is input alphabet
* Γ is stack symbols
* **δ** is the transition function: Q × (∑ ∪ {ε}) × S × Q × S\*
* **q0** is the initial state (q0 ∈ Q)
* Z0 is the initial stack top symbol (Z0 ∈ S)
* **F** is a set of accepting states (F ∈ Q)

A PDA for L = {set of all strings over [0,1] such that equal number of 0s and 1s, acceptance by final state.

For this, let us construct a PDA as;

P = {Q, Σ, Г, δ, q0, z0, F} be the PDA recognizing the given language. where, let us suppose

Q = {q0, q1, q2}

Σ = {0, 1}

Г = {0, 1, z0}

z0 = z0

q0 = q0

F = {q2}

Now δ is defined as;

1. δ(q0, Є, Є) = (q1, z0) //initialize the stack to indicate the bottom of stack.

2. δ(q1, 0, z0) = (q1, 0z0)

3. δ(q1, 1, z0) = (q1, 1z0)

4. δ(q1, 0, 0) = (q1, 00)

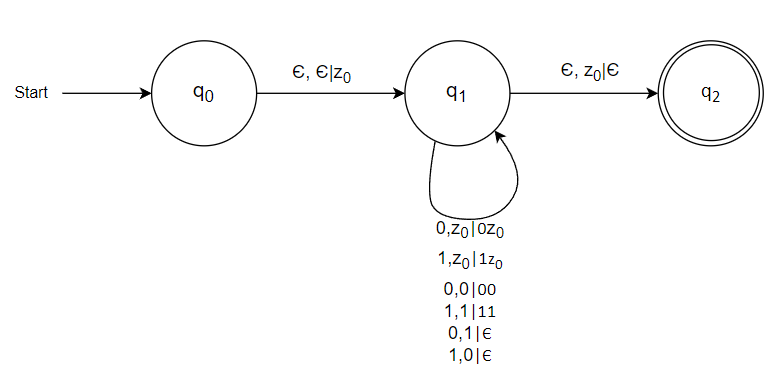
5. δ(q1, 1, 1) = (q1, 11)

6. δ(q1, 0, 1) = (q1, Є)

7. δ(q1, 1, 0) = (q1, Є)

8. δ(q1, Є, z0) = (q2, Є) // indicate the acceptance of string.

So, the graphical notation for the PDA constructed in example 1 can be constructed as;



**Source Code**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

enum State { q0, qf, qError };

State delta(State currentState, char inputSymbol, stack<char>& stack) {

switch (currentState) {

case q0:

if (inputSymbol == '0') {

stack.push(inputSymbol);

} else if (inputSymbol == '1') {

if (!stack.empty() && stack.top() == '0') {

stack.pop();

} else {

return qError;

}

} else {

return qError;

}

break;

case qf:

return qError;

case qError:

return qError;

}

return currentState;

}

bool isAccepted(const string& input) {

stack<char> stack;

State currentState = q0;

for (char symbol : input) {

currentState = delta(currentState, symbol, stack);

if (currentState == qError) {

return false;

}

}

return (currentState == q0 && stack.empty());

}

int main() {

string input;

cout << "Enter a string over {0,1}: ";

cin >> input;

if (isAccepted(input)) {

cout << "Accepted." << endl;

} else {

cout << "Rejected." << endl;

}

return 0;

}

**Output:**

Enter a string over {0,1}: 00110011

Accepted.

Enter a string over {0,1}: 1000101

Rejected.

**PRACTICAL NO.: 05**

**Title:**

Implementation and Analysis of TM for string recognition

**Objective:**

To Implement and Design the Tuning Machine (TM)

**Theory:**

Turing machine is an abstract machine developed by an English Mathematician Alan Turing in 1936. The model of computation provides a theoretical foundation for modern computers. A

Turing machine will have;

* A finite set of alphabets
* A finite set of states
* A linear tape which is potentially infinite to both end.

A Turing Machine T M is defined by the seven-tuples, M = (Q, Σ, Г, δ, q0, B, F) where,

* Q = the finite set of states of the finite control
* Σ = the finite set of input symbols
* Г = the complete set of tape symbols Σ is always subset of Г.
* q0 = the start state; q0 ε Q
* B = the blank symbol; B ε Г but B does not belong to Σ.
* F = the set of final or accepting states; F is subset of Q
* δ = the transition function defined by

Q × Г →Q × Г × (R, L, S); where R, L, S is the direction of movement of head left, or right or stationary. i.e. δ(q, x) = δ(p, Y, D); which means T M in state q and current tape symbol x, moves to next state P, replacing tape symbol x with Y and move the head either direction or remains at same cell of input tape.

A TM accepting the language [0n1n / n ≥ 1] over alphabet, ∑ = [0, 1].

Given finite sequence of 0’s and 1’s on tape and followed by blanks. The TM starts at state q0 and changes 0 to an X and moves to the right changing its state of q1.

At state q1, TM expects 1 and changes a 1 to Y and moves to the left changing the state to q2. If any number of 0s and Ys are seen, it remains on the state q1 and leaving these symbols unchanged and moving the head position to the right.

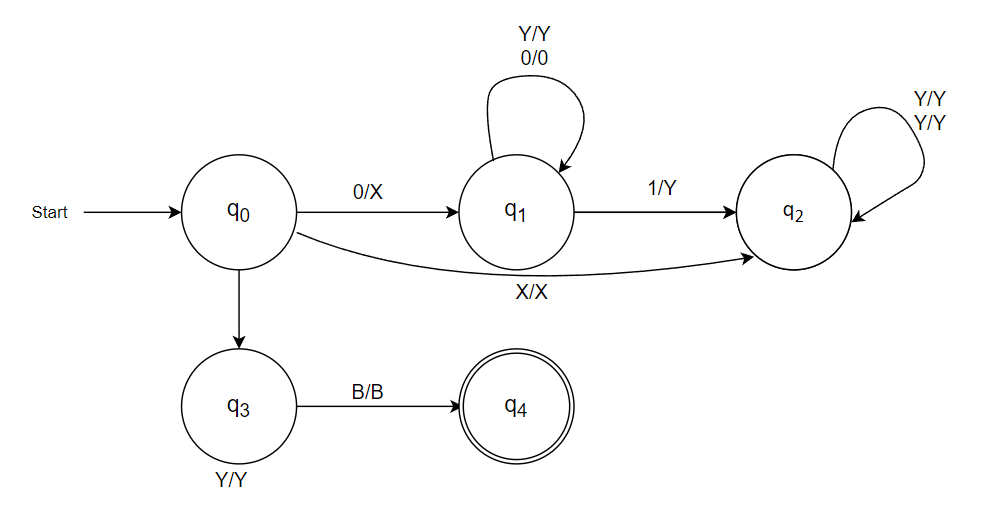
At state q2, if 0s or Y’s are seen, it leaves them as it is and moves to the left staying at the same state q2. If it sees X at state q2, the tape symbol is left unchanged moves to right switching its sate to q0.

At state q0, if it sees Y then the symbol is left unchanged and head is moved right changing to the state q3.

At sate q3, if Y is seen, it is left unchanged and head is moved to the right. If BLANK (here, ‘\0’) is seen at state q3, the string is accepted by switching the state to q4.

At any state, if the machine seen other than the defined symbols, it rejects the string.

The state transition diagram is shown in the figure below q3.



**Source Code:**

#include <iostream>

#include <vector>

using namespace std;

enum class State { q0, q1, q2, q3 };

int main() {

string input;

cout << "Enter a binary string: ";

cin >> input;

if (input.empty()) {

cout << "Error: Input string cannot be empty." << endl;

return 1;

}

vector<char> tape(input.size() + 2, 'B');

for (int i = 0; i < input.size(); i++) {

tape[i + 1] = input[i];

}

State state = State::q0;

int head = 1;

while (state != State::q3) {

char symbol = tape[head];

switch (state) {

case State::q0:

if (symbol == '0') {

tape[head] = 'X';

state = State::q1;

} else {

state = State::q3;

}

head++;

break;

case State::q1:

if (symbol == '0') {

tape[head] = 'X';

head++;

} else if (symbol == '1') {

state = State::q2;

} else {

state = State::q3;

}

break;

case State::q2:

if (symbol == '1') {

tape[head] = 'Y';

state = State::q1;

} else {

state = State::q3;

}

head--;

break;

default:

break;

}

}

int num\_x = 0, num\_y = 0;

for (char c : tape) {

if (c == 'X') {

num\_x++;

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

num\_y++;

}

}

if (state == State::q3 || num\_x != num\_y) {

cout << "String is not accepted." << endl;

} else {

cout << "String is accepted." << endl;

}

return 0;

}

**Output:**

Enter a binary string: 000111

String is accepted.

Enter a binary string: 000110

String is not accepted.