

TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

MADAN BHANDARI MEMORIAL COLLEGE

New Baneshwor, Kathmandu

**Lab Report of Theory of Computation**

**Submitted by:** **Submitted to:**

Name: Sudip Pradhan Department of B.Sc. CSIT

Symbol No.: 29170

Semester : Fourth

Signature



**Madan Bhandari Memorial College**

**Department of Computer Science and information and technology**

**Binayaknagar, New Baneshwor, Kathmandu**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name:**  Sudip Pradhan | **Semester :** 4th | **Batch :**2078 | **Subject: TOC** | **Symbol No. :**  29170 |

**Practical Record Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Program No.** | **Title of Programs** | **Date of**  **Submission** | **Signature** |
| **1.** | Program to construct a DFA that accepts the language L = {an | n >= 1}. |  |  |
| **2.** | Program to construct a DFA which accepts the language L = {an bm | n mod 2 = 0, m >= 1}. |  |  |
| **3.** | Program to construct a DFA which accepts the language L = {an bm | n mod 2 = 0, m >= 1}. |  |  |
| **4.** | Program to construct a DFA that accepts odd number of 0's and odd number of 1's over the characters {0, 1}. |  |  |
| **5.** | : Program to construct a NFA that accepts strings containing the substring ‘101’ |  |  |
| **6.** | Program to construct a NFA that accepts strings ending with ‘01’. |  |  |
| **7.** | Program to construct a NFA that accepts strings starting with ‘10’. |  |  |
| **8.** | Program to convert NFA to DFA. |  |  |
| **9.** | Introduction to Perl Programming Language. |  |  |
| **10.** | Perl Programs to accept strings:   1. starting with ‘a’. 2. starting with ‘a’ and ending with ‘b’. 3. having substring ‘101’. 4. of the form an bn . 5. of the form an b2n . |  |  |

# Madan Bhandari Memorial College

# 

# Lab No.: 1

# TITLE: WAP TO CONSTRUCT DFA THAT ACCEPTS THE A LANGUAGE L = {A^N | N >= 1}

**SOURCE CODE:**

#include <stdio.h>

#include <string.h>

#define NUM\_STATES 2

#define ALPHABET\_SIZE 1

// DFA transition table

int transitionTable[NUM\_STATES][ALPHABET\_SIZE] = {

{1}, // From state 0, on input 'a', transition to state 1

{1} // From state 1, on input 'a', remain in state 1 (loop)

};

// DFA accepting states

int acceptingStates[] = {1}; // Only state 1 is an accepting state

// DFA accepting function

int isAccepted(char\* input) {

int currentState = 0;

int i = 0;

while (input[i] != '\0') {

int inputIndex = input[i] - 'a'; // Mapping input character to index

if (inputIndex < 0 || inputIndex >= ALPHABET\_SIZE)

return 0; // Invalid input character

currentState = transitionTable[currentState][inputIndex];

i++;

}

// Check if the final state is an accepting state

int j;

for (j = 0; j < sizeof(acceptingStates) / sizeof(acceptingStates[0]); j++) {

if (currentState == acceptingStates[j])

return 1; // Accepted

}

return 0; // Not accepted

}

int main() {

char input[100];

printf("Enter a string: ");

scanf("%s", input);

if (isAccepted(input))

printf("Accepted");

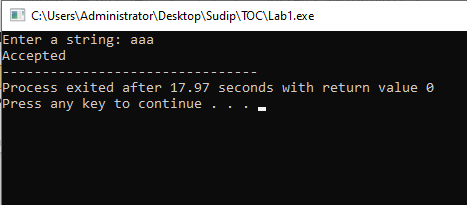
else

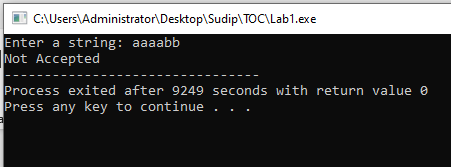
printf("Not Accepted");

return 0;

}

**OUTPUT**

****

****



# 

# Lab No.: 2

# TITLE: PROGRAM TO CONSTRUCT A DFA WHICH ACCEPTS THE LANGUAGE L = {ANBM| N MOD 2 = 0, M >=1}

**SOURCE CODE:**

#include <stdio.h>

#include <stdbool.h>

// DFA transition function

int transition(int state, char input) {

switch(state) {

case 0:

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

else if (input == 'b') return 2;

else return -1; // Invalid transition

case 1:

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

else if (input == 'b') return 2;

else return -1; // Invalid transition

case 2:

if (input == 'b') return 2;

else return -1; // Invalid transition

}

return -1; // Invalid state

}

// Function to check if the input string is accepted by the DFA

bool isAccepted(char \*input) {

int currentState = 0;

int aCount = 0;

int bCount = 0;

while (\*input != '\0') {

currentState = transition(currentState, \*input);

if (currentState == -1) return false; // Invalid transition

if (\*input == 'a') {

aCount++;

} else if (\*input == 'b') {

bCount++;

// Ensure 'b' does not appear before 'a'

if (currentState == 0) return false;

}

input++;

}

// Check if the final state is an accepting state (state 2)

// and if aCount is even and bCount is at least 1

return currentState == 2 && (aCount % 2 == 0) && bCount >= 1;

}

int main() {

char input[100];

printf("Enter the input string: ");

scanf("%s", input);

if (isAccepted(input))

printf("Accepted");

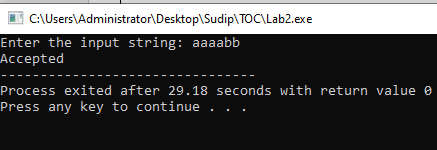
else

printf("Not Accepted");

return 0;

}

**OUTPUT**



# 

# Madan Bhandari Memorial College

# 

# Lab No.: 3

# TITLE: WAP TO CONSTRUCT A DFA THAT ACCEPTS THE STRINGS ENDING WITH ‘01’ OVER THE CHARACTERS {0, 1}

**SOURCE CODE:**

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#define STATES 3

#define ALPHABET 2

// DFA Transition Table

int transitionTable[STATES][ALPHABET] = {

{1, 0}, // State 0

{1, 2}, // State 1

{1, 0} // State 2 (final state)

};

// Function to check if a given string is accepted by the DFA

bool isAccepted(char \*string) {

int currentState = 0; // Start from the initial state

int len = strlen(string);

int i;

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

if (string[i] != '0' && string[i] != '1') // Check if the input character is valid

return false;

int inputSymbol = string[i] - '0'; // Convert char to integer

currentState = transitionTable[currentState][inputSymbol]; // Move to the next state based on the input symbol

}

// Check if the final state is reached

return currentState == 2;

}

int main() {

char string[100];

printf("Enter a string: ");

scanf("%s", string);

if (isAccepted(string))

printf("String is accepted by the DFA.\n");

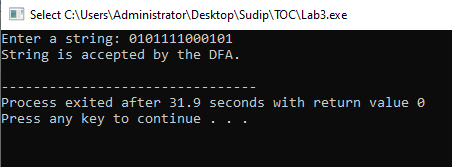
else

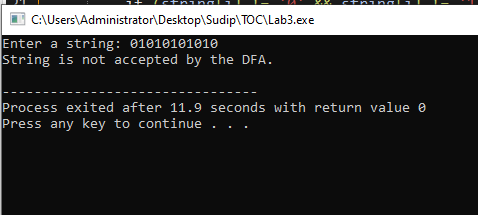
printf("String is not accepted by the DFA.\n");

return 0;

}

**OUTPUT**





.



# 

# Lab No.: 4

# WAP TO CONSTRUCT A DFA THAT ACCEPT ODD NUMBER OF ‘0’S AND ODD NUMBERS OF 1’S OVER THE CHARACTERS {0, 1}

**SOURCE CODE**

#include <stdio.h>

#include <stdbool.h>

#define STATES 3 // Number of states

#define ALPHABET\_SIZE 2 // Alphabet size

// DFA transition table

int transitionTable[STATES][ALPHABET\_SIZE] = {

{1, 2}, // State 0

{0, 2}, // State 1

{2, 1} // State 2 (final state)

};

// Function to check if the string is accepted by the DFA

bool isAccepted(char \*input) {

int currentState = 0;

int count0 = 0, count1 = 0;

int i = 0;

// Iterate through the input string

while (input[i] != '\0') {

// Get the input symbol

char symbol = input[i] - '0';

// Update the count of 0's and 1's

if (symbol == 0) {

count0++;

} else {

count1++;

}

// Update the current state using the transition table

currentState = transitionTable[currentState][symbol];

// Move to the next symbol in the input string

i++;

}

// Check if the final state is reached and it's an accepting state

return currentState == 2 && count0 % 2 == 1 && count1 % 2 == 1;

}

int main() {

char input[100];

printf("Enter the input string (containing only 0s and 1s): ");

scanf("%s", input);

// Check if the input string is accepted

if (isAccepted(input)) {

printf("String \"%s\" is accepted by the DFA.\n", input);

} else {

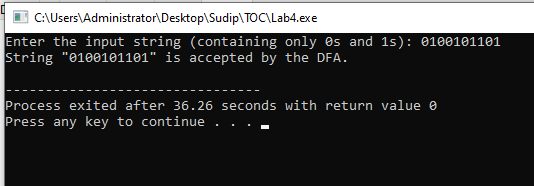
printf("String \"%s\" is not accepted by the DFA.\n", input);

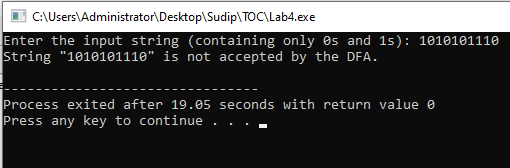
}

return 0;

}

**OUTPUT**





# Madan Bhandari Memorial College

# Lab No.: 5

# TITLE: PROGRAM TO CONSTRUCT A NFA THAT ACCEPTS STRINGS CONTAINING THE SUBSTRING ‘101’.

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

*// Define the NFA as a set of states and transitions*

vector<int> states = {0, 1, 2, 3}; // States are represented by integers (0, 1, 2, ...)

vector<vector<pair<char, int>>> transitions = {

{{'0', 0}, {'1', 0}, {'1', 1}},

{{'0', 2}},

{{'1', 3}},

{{'0', 3}, {'1', 3}}}; // Transitions are represented by pairs of characters and states (character, state)

*// Define a function to simulate the NFA on a given string*

bool simulate\_nfa(string input)

{

// Start at the initial state (state 0)

vector<int> current\_states = {0};

*// Loop through each character in the input string*

for (char c : input)

{

*// Find all possible transitions from the current states for the current character*

vector<int> next\_states;

for (int state : current\_states)

{

for (auto transition : transitions[state])

{

if (transition.first == c)

{

next\_states.push\_back(transition.second);

}

}

}

*// If there are no possible transitions, the input string is not accepted*

if (next\_states.empty())

{

return false;

}

*// Update the current states to the next states*

current\_states = next\_states;

}

*// If the final state is an accepting state, the input string is accepted*

for (int state : current\_states)

{

if (state == 3)

{

return true;

}

}

return false;

}

*// Define the main function to run the program*

int main()

{

// Get input from the user

string input;

cout << "Enter a string to check: ";

cin >> input;

*// Simulate the NFA on the input string and output the result*

if (simulate\_nfa(input))

{

cout << "String contains substring 101." << endl;

}

else

{

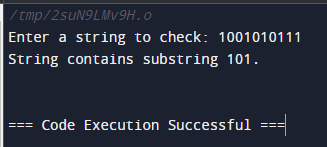
cout << "String does not contain substring 101." << endl;

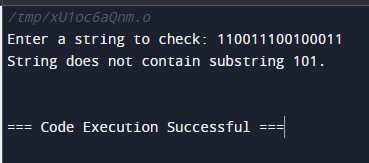
}

return 0;

}

**OUTPUT**

****

****

# Madan Bhandari Memorial College

# 

# Lab No.: 6

# TITLE: WRITE A PROGRAM TO CONSTRUCT NFA THAT ACCEPTS STRING ENDING WITH ‘01’

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

*// Define the NFA as a set of states and transitions*

vector<int> states = {0, 1, 2}; // States are represented by integers (0, 1, 2, ...)

vector<vector<pair<char, int>>> transitions = {

{{'0', 0}, {'1', 0}, {'0', 1}},

{{'1', 2}},

{{}}}; // Transitions are represented by pairs of characters and states (character, state)

*// Define a function to simulate the NFA on a given string*

bool simulate\_nfa(string input)

{

*// Start at the initial state (state 0)*

vector<int> current\_states = {0};

*// Loop through each character in the input string*

for (char c : input)

{

*// Find all possible transitions from the current states for the current character*

vector<int> next\_states;

for (int state : current\_states)

{

for (auto transition : transitions[state])

{

if (transition.first == c)

{

next\_states.push\_back(transition.second);

}

}

}

*// If there are no possible transitions, the input string is not accepted*

if (next\_states.empty())

{

return false;

}

// *Update the current states to the next states*

current\_states = next\_states;

}

*// If the final state is an accepting state, the input string is accepted*

for (int state : current\_states)

{

if (state == 2)

{

return true;

}

}

return false;

}

*// Define the main function to run the program*

int main()

{

*// Get input from the user*

string input;

cout << "Enter a string to check: ";

cin >> input;

*// Simulate the NFA on the input string and output the result*

if (simulate\_nfa(input))

{

cout << "String ends with 01." << endl;

}

else

{

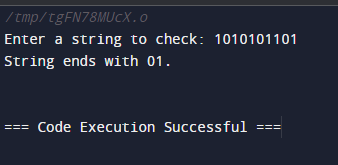
cout << "String does not end with 01." << endl;

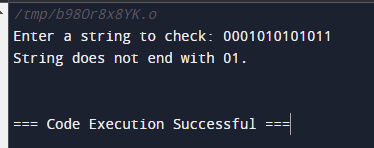
}

return 0;

}

**OUTPUT**

****

****

# Madan Bhandari Memorial College

# 

# Lab No.: 7

# TITLE: WAP TO CONSTRUCT A NFA THAT ACCEPTS STRINGS STARTING WITH ‘10’.

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

// Define the NFA as a set of states and transitions

vector<int> states = {0, 1, 2}; // States are represented by integers (0, 1, 2, 3, ...)

vector<vector<pair<char, int>>> transitions = {

{{'1', 1}},

{{'0', 2}},

{{'0', 2}, {'1', 2}},

{{}}}; // Transitions are represented by pairs of characters and states (character, state)

*I*

bool simulate\_nfa(string input)

{

*// Start at the initial state (state 0)*

vector<int> current\_states = {0};

*// Loop through each character in the input string*

for (char c : input)

{

// Find all possible transitions from the current states for the current character

vector<int> next\_states;

for (int state : current\_states)

{

for (auto transition : transitions[state])

{

if (transition.first == c)

{

next\_states.push\_back(transition.second);

}

}

}

*// If there are no possible transitions, the input string is not accepted*

if (next\_states.empty())

{

return false;

}

*// Update the current states to the next states*

current\_states = next\_states;

}

*// If the final state is an accepting state, the input string is accepted*

for (int state : current\_states)

{

if (state == 2)

{

return true;

}

}

return false;

}

*// Define the main function to run the program*

int main()

{

*// Get input from the user*

string input;

cout << "Enter a string to check: ";

cin >> input;

*// Simulate the NFA on the input string and output the result*

if (simulate\_nfa(input))

{

cout << "String starts with 10." << endl;

}

else

{

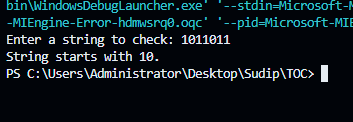
cout << "String does not start with 10." << endl;

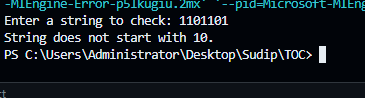
}

return 0;

}

**OUTPUT**







# 

# Lab No.: 8

# TITLE: PROGRAM TO CONVERT NFA TO DFA

**PROGRAM CODE:**

#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("Follow 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: ");

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("\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\n");

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\n");

else

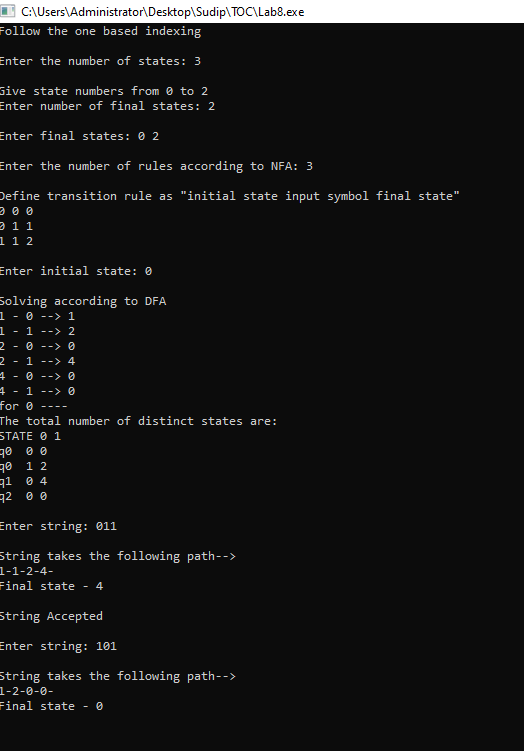
printf("\nString Rejected\n");

}

return 0;

}

**OUTPUT**

****



# Lab No.: 9

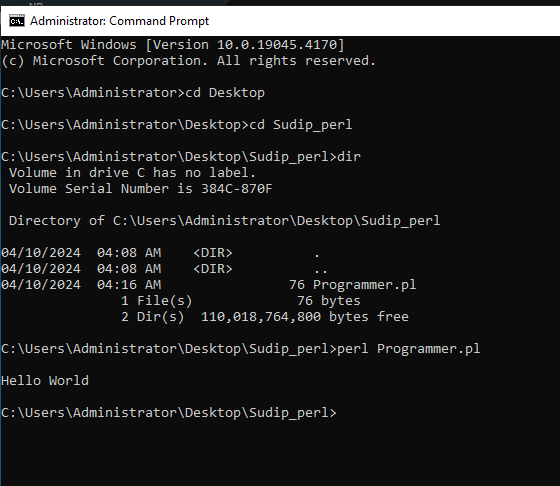
# TITLE:INTRODUCTION TO PERL PROGRAMMING LANGUAGE

**INTRODUCTIONS:**

Perl is a highly capable and feature-rich programming language that has been developed for over 36 years. It is a general-purpose language that supports both procedural and object-oriented programming paradigms. Perl is known for its extensive library of over 25,000 extension modules and a large developer community. It is widely used for a variety of tasks, including system administration, web development, network programming, and more.

# my first program

print "\nHello World\n";



# variables in Perl

my $age = 22; # integer

my $name = "Sudip Pradhan"; # string

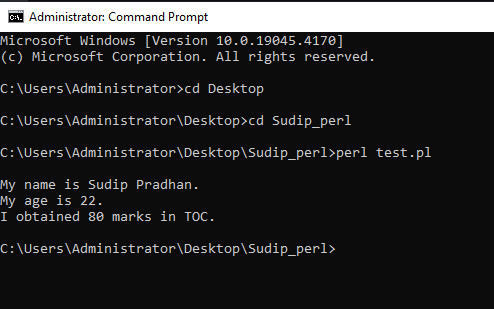
my $marks = 80; # floating point

print "\nMy name is $name.\n";

print "My age is $age.\n";

print "I obtained $marks marks in TOC.\n";

**OUTPUT:**



# example of pattern matching

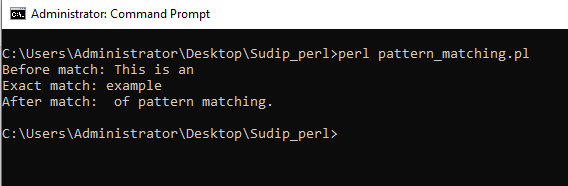
$string = "This is an example of pattern matching.";

$string =~ m/example/; # binding operator

print "Before match: $`\n"; # string preceding a successful pattern match

print "Exact match: $&\n"; # substring that matched the pattern

print "After match: $'\n" # string after the successful pattern match



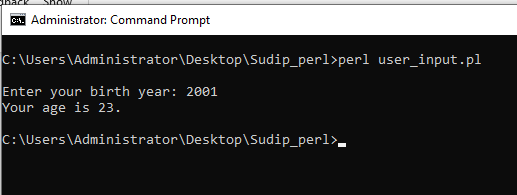
# example of user input

print "\nEnter your birth year: ";

$year = <STDIN>;

$age = 2024 - $year;

print "Your age is $age.\n"





# Lab No.: 10

**TITLE: USE LIBRARY TOOLS LIKE NLTK TO SPLIT THE WORDS OF A SENTENCE**

. **LAB 10: Perl Programs to accept strings:**

1. **starting with ‘a’.**
2. **starting with ‘a’ and ending with ‘b’.**
3. **having substring ‘101’.**
4. **of the form an bn .**
5. **of the form an b2n .**
6. starting with ‘a’.

# accept strings starting with a

print "\nEnter your string: ";

$string = <STDIN>;

if($string =~ /^a/){

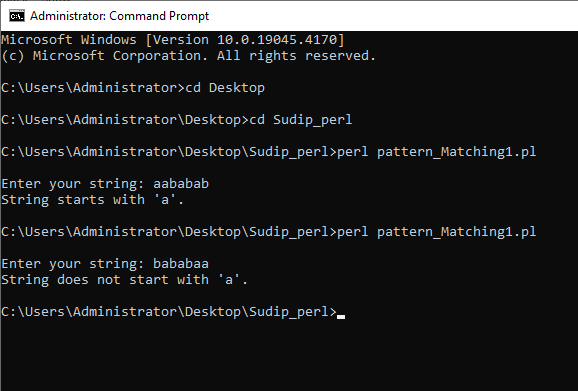
print "String starts with 'a'.\n"

} else {

print "String does not start with 'a'.\n"

}

OUTPUT



1. starting with ‘a’ and ending with ‘b’.

# accept strings starting with a and ending with b

print "\nEnter your string: ";

$string = <STDIN>;

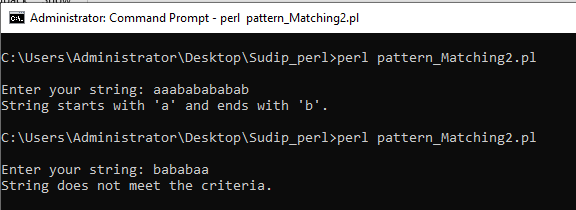
if($string =~ /^a.\*b/){

print "String starts with 'a' and ends with 'b'.\n"

} else {

print "String does not meet the criteria.\n"

}



1. having substring ‘101’.

# accept strings having substring 101

print "\nEnter your string: ";

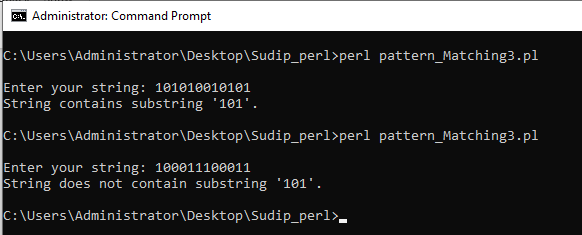
$string = <STDIN>;

if($string =~ /.\*101.\*/){

print "String contains substring '101'.\n"

} else {

print "String does not contain substring '101'.\n"

}

1. of the form an bn .

# accept strings of the form a^nb^n

print "\nEnter your string: ";

$string = <STDIN>;

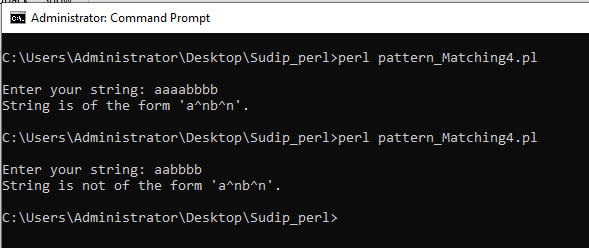
if($string =~ /^(a(?1)?b)$/){

print "String is of the form 'a^nb^n'.\n"

} else {

print "String is not of the form 'a^nb^n'.\n"

}



1. of the form an b2n .

# accept strings of the form a^nb^2n

print "\nEnter your string: ";

$string = <STDIN>;

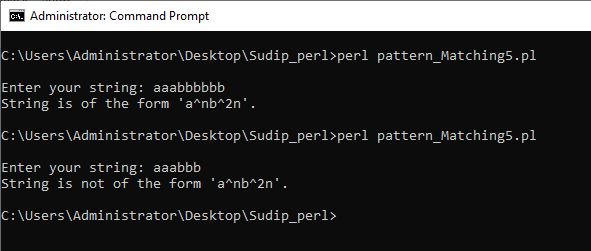
if($string =~ /^(a(?1)?bb)$/){

print "String is of the form 'a^nb^2n'.\n"

} else {

print "String is not of the form 'a^nb^2n'.\n"

}



THE END