**TOC (CSA1388)-LAB EXPERIMENTS:**

1.Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with a and end with a.

CODE:

#include <stdio.h>

// DFA transition function

int transition(int state, char input) {

if (state == 0 && input == 'a') return 1; // Move to state 1 on input 'a'

if (state == 1 && input == 'a') return 1; // Stay in state 1 on input 'a'

return 0; // Any other input leads to a non-accepting state

}

int main() {

int current\_state = 0;

char input\_string [100];

printf("Enter an input string: ");

scanf("%s", input\_string);

for (int i = 0; input\_string[i] != '\0'; i++) {

current\_state = transition(current\_state, input\_string[i]);

}

if (current\_state == 1) {

printf("String accepted\n");

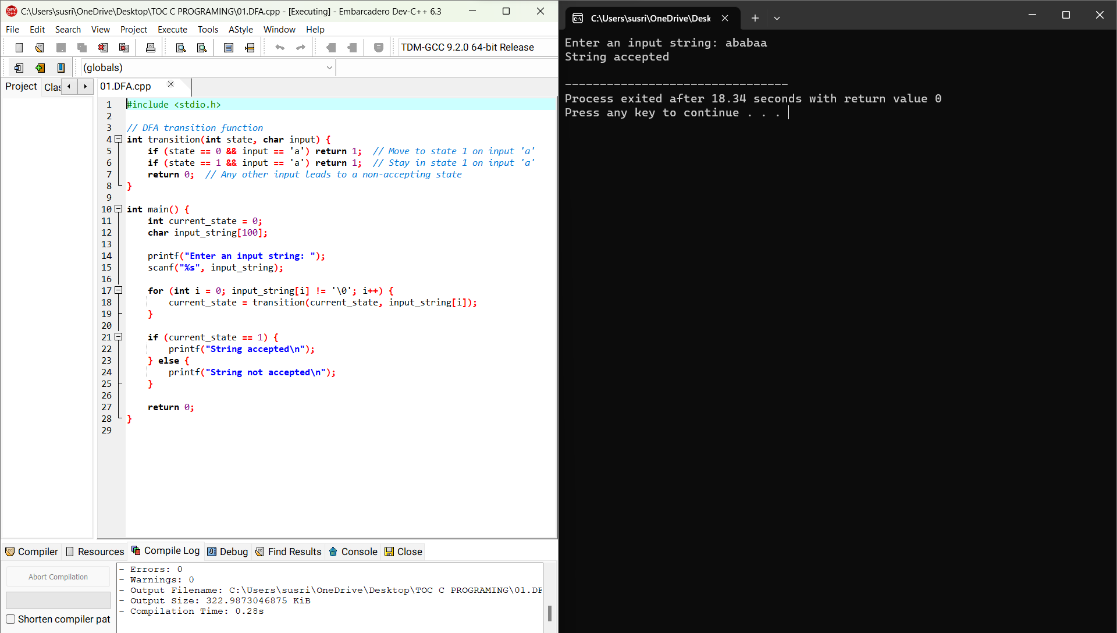
} else {

printf("String not accepted\n");

}

return 0; }

OUTPUT:



1. Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with 0 and end with 1.

CODE:

#include <stdio.h>

#include <stdbool.h>

// DFA transition function

int transition(int state, char input) {

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

if (state == 1 && input == '1') return 2;

return -1; // Invalid transition

}

// DFA simulation function

bool simulateDFA(const char \*input) {

int currentState = 0;

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

currentState = transition(currentState, input[i]);

if (currentState == -1) {

return false; // Invalid transition, input string not accepted

}

}

return currentState == 2; // Check if the final state is reached

}

int main() {

char input[100];

printf("Enter a string: ");

scanf("%s", input);

if (simulateDFA(input)) {

printf("Accepted\n");

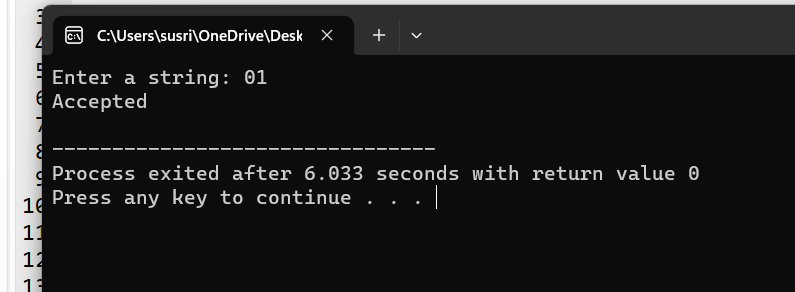
} else {

printf("Not Accepted\n");

return 0;

}

OUTPUT:



1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0A1 A → 0A | 1A | ε

CODE:

#include <stdbool.h>

#include <string.h>

// Function to check if the given string belongs to the language

bool belongsToLanguage(char \*str, int start, int end) {

if (start > end) {

return true; // Base case: ε production

}

if (str[start] == '0' && str[end] == '1') {

for (int i = start + 1; i < end; i++) {

if (belongsToLanguage(str, start + 1, i) && belongsToLanguage(str, i + 1, end - 1)) {

return true;

}

}

}

return false;

}

int main() {

char input[100];

printf("Enter a string: ");

scanf("%s", input);

int length = strlen(input);

if (belongsToLanguage(input, 0, length - 1)) {

printf("Doesn't Belongs to the language.\n");

} else {

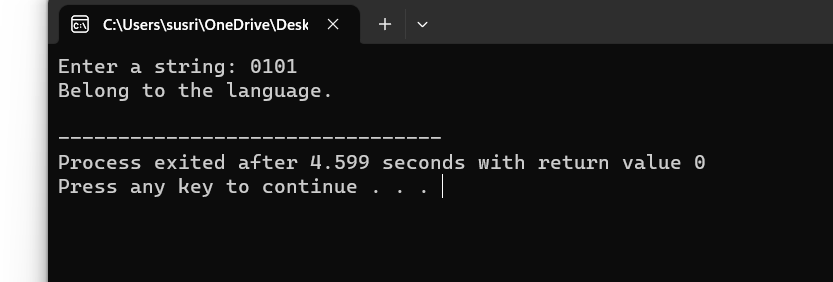
printf("Belong to the language.\n");

}

return 0;

}

OUTPUT:



1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S0 | 1S1 | 0 | 1 | ε.

CODE:

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// Function to check if a string belongs to the CFG language

bool belongsToCFG(char \*input, int start, int end) {

// Base case: empty string

if (start > end) {

return true;

}

// Case 1: S -> 0S0

if (input[start] == '0' && input[end] == '0') {

for (int i = start + 1; i < end; i++) {

if (belongsToCFG(input, start + 1, i - 1) && belongsToCFG(input, i + 1, end - 1)) {

return true;

}

}

}

// Case 2: S -> 1S1

if (input[start] == '1' && input[end] == '1') {

for (int i = start + 1; i < end; i++) {

if (belongsToCFG(input, start + 1, i - 1) && belongsToCFG(input, i + 1, end - 1)) {

return true;

}

}

}

return false;

}

int main() {

char input\_string[100];

printf("Enter an input string: ");

scanf("%s", input\_string);

int len = strlen(input\_string);

if (belongsToCFG(input\_string, 0, len - 1)) {

printf("String belongs to the CFG language.\n");

} else {

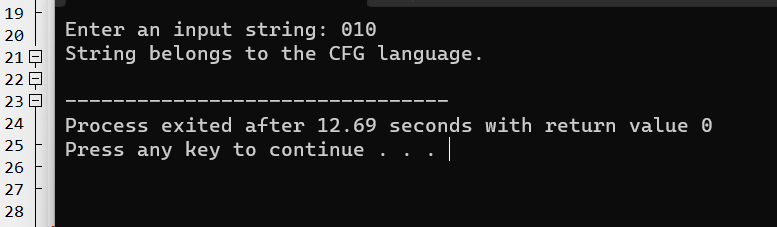
printf("String does not belong to the CFG language.\n");

}

return 0;

}

OUTPUT:



1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S0 | A A → 1A | ε

CODE:

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// Function to check if a string belongs to the CFG language

bool belongsToCFG(char \*str) {

if (\*str == '\0') {

return true; // A -> e

}

if (\*str == '0' && str[strlen(str) - 1] == '0') {

return belongsToCFG(str + 1) && belongsToCFG(str + 1); // S -> 0S0

}

if (\*str == '1') {

return belongsToCFG(str + 1); // A -> 1A

}

return false;

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

if (belongsToCFG(str)) {

printf("The string belongs to the language defined by the CFG.\n");

} else {

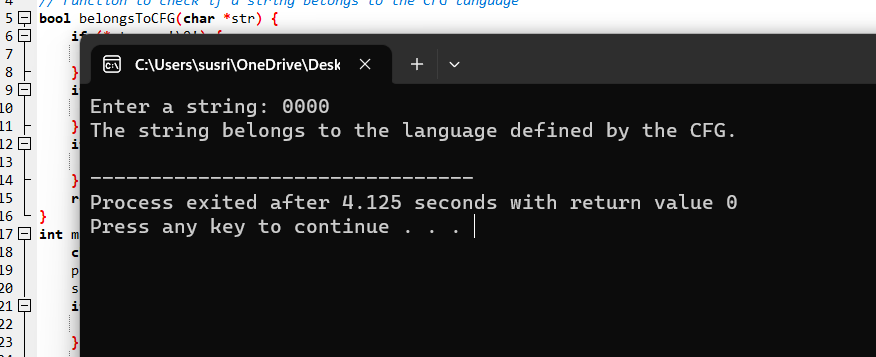
printf("The string does not belong to the language defined by the CFG.\n");

}

return 0;

}

OUTPUT:



1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S1 | ε

CODE:

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// Function to check if the given string belongs to the language defined by the CFG

bool belongsToLanguage(const char \*str, int start, int end) {

if (start > end) {

return true; // e production

}

if (str[start] == '0' && str[end] == '1') {

for (int i = start + 1; i < end; i++) {

if (belongsToLanguage(str, start + 1, i - 1) && belongsToLanguage(str, i + 1, end - 1)) {

return true;

}}} return false; }

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

int length = strlen(str);

if (belongsToLanguage (str, 0, length - 1)) {

printf("The string belongs to the language defined by the CFG.\n");

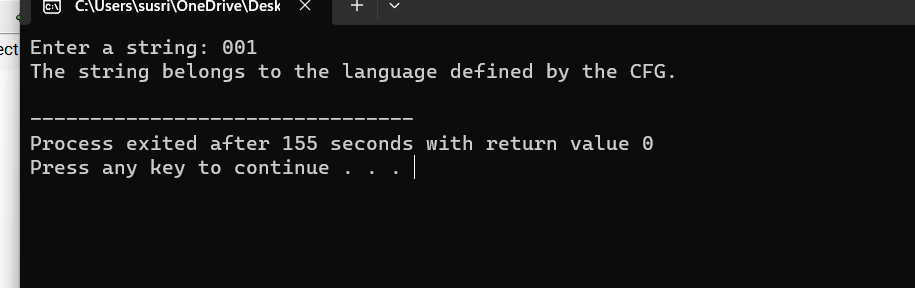
} else {

Printf("The string does not belong to the language defined by the CFG.\n");

}

return 0;}

OUTPUT:



1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → A101A, A → 0A | 1A | ε

CODE:

#include <stdio.h>

#include <string.h>

// Function to check if a string belongs to language S

int isBelongToLanguageS(char \*str, int start, int end) {

// Base case: empty string is accepted

if (start > end) {

return 1;

}

// Apply production rules for A

if (str[start] == '0') {

return isBelongToLanguageS(str, start + 1, end);

}

if (str[start] == '1') {

return isBelongToLanguageS(str, start + 1, end);

}

return 0; // Not a valid string for language S

}

// Function to check if a string belongs to language A

int isBelongToLanguageA(char \*str, int start, int end) {

if (start > end) {

return 1; // Empty string is accepted

}

if (str[start] == '0' || str[start] == '1') {

return isBelongToLanguageA(str, start + 1, end);

}

return 0; // Not a valid string for language A

}

int main() {

char input[100];

printf("Enter a string: ");

scanf("%s", input);

int len = strlen(input);

if (len >= 5 && input[len - 4] == '1' && input[len - 3] == '0' && input[len - 2] == '1' && input[len - 1] == '1') {

if (isBelongToLanguageA(input, 0, len - 5) && isBelongToLanguageA(input, len - 5, len - 5)) {

printf("The string not belongs to the language defined by the CFG.\n");

} else {

printf("The string belong to the language defined by the CFG.\n");

}

} else {

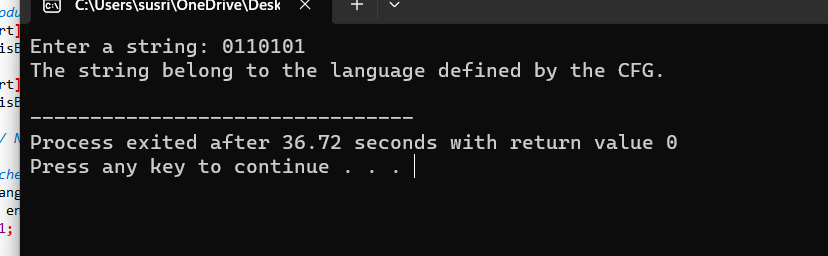
printf("The string belong to the language defined by the CFG.\n");

}

return 0;

}

OUTPUT:



1. Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given language representing strings that start with b and end with a

CODE:

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// Number of states in the NFA

#define NUM\_STATES 3

// Transition function for the NFA

int transitions[NUM\_STATES][2][2] = {

{{0, 1}, {-1, -1}}, // State 0

{{-1, -1}, {2, -1}}, // State 1

{{-1, -1}, {-1, -1}} // State 2 (accepting state)

};

// Function to simulate NFA for the given input string

bool simulateNFA(const char \*input) {

int currentState = 0;

int inputLength = strlen(input);

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

char c = input[i];

int inputIndex = (c == 'b') ? 0 : ((c == 'a') ? 1 : -1);

if (inputIndex == -1) {

// Invalid input character

return false;

}

int nextState1 = transitions[currentState][inputIndex][0];

int nextState2 = transitions[currentState][inputIndex][1];

if (nextState1 == -1 && nextState2 == -1) {

// No valid transitions for this input character

return false;

}

currentState = nextState1;

if (nextState2 != -1) {

currentState = nextState2;

}

}

return currentState == 2; // Check if the final state is the accepting state

}

int main() {

char input[100];

printf("Enter a string: ");

scanf("%s", input);

if (simulateNFA(input)) {

printf("Accepted\n");

} else {

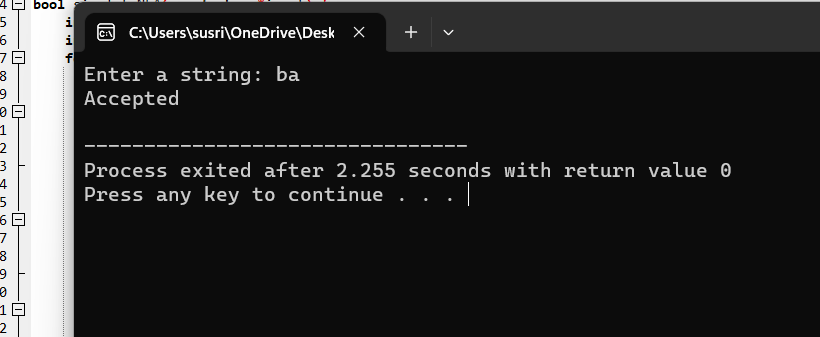
printf("Rejected\n");

}

return 0;

}

OUTPUT:



1. Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with o and end with 1.

CODE:

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

int transition(int state, char symbol) {

if (state == 0 && symbol == '0')

return 1;

else if (state == 1 && symbol == '1')

return 2;

return -1; // Invalid transition

}

bool simulateNFA(const char \*input) {

int currentState = 0;

int inputLength = strlen(input);

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

currentState = transition(currentState, input[i]);

if (currentState == -1)

return false;

}

return currentState == 2;

}

int main() {

char input[100];

printf("Enter a string: ");

scanf("%s", input);

if (simulateNFA(input)) {

printf("Accepted: String follows the pattern.\n");

} else {

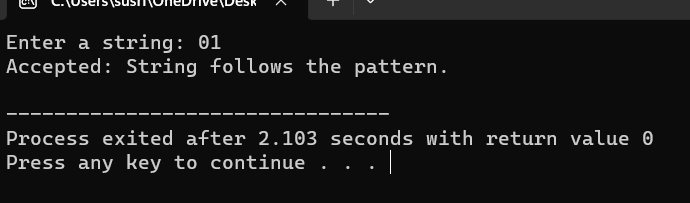
printf("Rejected: String does not follow the pattern.\n");

}

return 0;

}

OUTPUT:



1. Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

CODE:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_STATES 50

#define MAX\_TRANSITIONS 50

typedef struct {

int transitions[MAX\_TRANSITIONS];

int transitionCount;

} State;

State nfa[MAX\_STATES];

bool visited[MAX\_STATES];

void epsilonClosure(int state) {

if (visited[state]) return;

visited[state] = true;

printf("%d ", state);

for (int i = 0; i < nfa[state].transitionCount; i++) {

int nextState = nfa[state].transitions[i];

epsilonClosure(nextState);

}

}

int main() {

int numStates, numSymbols;

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

scanf("%d", &numStates);

printf("Enter the number of symbols (excluding e): ");

scanf("%d", &numSymbols);

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

nfa[i].transitionCount = 0;

printf("Enter the number of transitions for state %d: ", i);

int numTransitions;

scanf("%d", &numTransitions);

printf("Enter the transitions (state indices) for state %d: ", i);

for (int j = 0; j < numTransitions; j++) {

scanf("%d", &nfa[i].transitions[j]);

nfa[i].transitionCount++;

}

}

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

printf("e-closure(%d): ", i);

for (int j = 0; j < MAX\_STATES; j++) {

visited[j] = false;

}

epsilonClosure(i);

printf("\n");

}

return 0;

}

OUTPUT:

