Lab Report for Theory of Computation

LAB1 :Write a Program to Find Prefixes, Suffixes, and Substrings from a Given String

1. **Objective**

The objective of this lab is to write a C program that accepts a string from the user and generates all possible prefixes, suffixes, and substrings of the given string.

# Theory

A string is a sequence of characters. In this context:

Prefixes: These are substrings that begin at the start of the string and increase in length one character at a time.

Suffixes: These are substrings that end at the end of the string and increase in length by moving from the end to the start

Substrings: These are any sequence of consecutive characters found within the string

For example, given the string "abc", the possible prefixes, suffixes, and substrings would be: Prefixes: a, ab, abc

Suffixes: abc, bc, c

Substrings: a, b, c, ab, bc, abc

1. **Source Code**

#include <stdio.h>

#include <string.h>

// Function to print all prefixes

void printPrefixes(char str[])

{

printf("Prefixes:\n");

for (int i = 1; i <= strlen(str); i++)

{

for (int j = 0; j < i; j++)

{

printf("%c", str[j]);

}

printf("\n");

}

}

// Function to print all suffixes

void printSuffixes(char str[])

{

printf("\nSuffixes:\n");

for (int i = 0; i < strlen(str); i++)

{

for (int j = i; j < strlen(str); j++) {

printf("%c", str[j]);

}

printf("\n");

}

}

void printSubstrings(char str[])

{

printf("\nSubstrings:\n")

int len = strlen(str);

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

{

for (int j = i; j < len; j++)

{

for (int k = i; k <= j; k++)

{

printf("%c", str[k]);

}

printf("\n");

}

}

}

int main() {

char str[100];

// Asking the user to enter the string

printf("Enter a string: ");

scanf("%s", str);

// Calling functions to print

printPrefixes(str);

printSuffixes(str);

printSubstrings(str);

return 0;

}

# Output

****

1. **Conclusion**

In this lab, we successfully implemented a C program to find and display all prefixes, suffixes, and substrings of a given string. The user inputs a string, and the program efficiently processes the input to generate and display the desired results. This experiment helps understand string manipulation and related operations in theory of computation.

**LAB 2: Implementation of the string that starts with "01" using a DFA.**

1. **Objective**

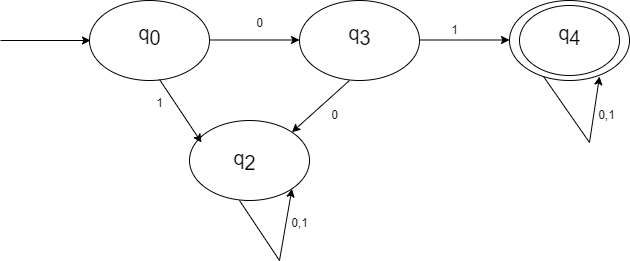
To implement a program that checks whether a given binary string starts with "01" using a DFA.

# Theory

A Deterministic Finite Automaton (DFA) is a theoretical model of computation that consists of a set of states, an alphabet, and a transition function. For this lab, the DFA will be designed to

recognize binary strings (numbers) that start with "01". The DFA accepts or rejects the string based on the input, ensuring it begins with the required prefix.

Transition Diagram:



# Source Code

#include <stdio.h> #include <string.h>

// DFA implementation to check if a binary string starts with "01"

int dfa(char str[]) {

int state = 0; // Start state q0

for (int i = 0; i < strlen(str); i++) {

char input = str[i];

switch (state) {

case 0:

if (input == '0') { state = 1;}

else {

state = -1;}

break;

case 1

if (input == '1') { state = 2;} else {

state = -1;}

break;

case 2:

break; default

return 0;

}

}

if (state == 2)

{

return 1

}

else

{

return 0;

}

}

int main() {

char str[100];

printf("Enter a binary number (a string of 0s and 1s): "); scanf("%s", str);

if (dfa(str)) {

printf("Accepted: The number starts with '01'.\n");

} else {

printf("Rejected: The number does not start with '01'.\n");

}

return 0;

}

# Output

# 

1. **Conclusion**

In this lab, we successfully implemented a DFA to check if binary strings start with "01". The program accurately evaluates the input string and provides feedback on its acceptance based on the defined criteria.

**LAB 3: Implementation of the string that ends with "01" using a DFA.**

1. **Objective**

To implement a program that checks whether a given binary string ends with "01" using a DFA

1. **Theory**

A Deterministic Finite Automaton (DFA) is a theoretical model used to recognize patterns within input strings. A DFA consists of a finite number of states, including an initial state and one or more accepting states. In this lab, we implement a DFA that accepts binary strings composed of 0 and 1, specifically those that end with the sequence "01". The DFA transitions between states based on the input characters, ultimately determining if the string meets the acceptance criteria.

Transition Table:

|  |  |  |
| --- | --- | --- |
| δ | 0 | 1 |
| → q0 | q1 | q0 |
| q1 | q2 | q0 |
| q2 | q2 | qf |
| \*qf | qf | qf |

1. **Source Code**

#include <stdio.h> #include <string.h>

int isAccepted(char \*str) {

int state = 0;

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

char ch = str[i]; switch (state) {

case 0:

if (ch == '0') { state = 1;}

break; case 1:

if (ch == '1') { state = 2;}

break; case 2:

if (ch == '0') {state= 1;

}

else if (ch == '1') { state = 2;}

break;

default:

return 0;

}

}

return (state==2)}

int main() {

char input[100];

printf("Enter a binary string (consisting of 0 and 1): "); scanf("%s", input);

if (isAccepted(input)) {

printf("Accepted: The string ends with '01'.\n");

} else {

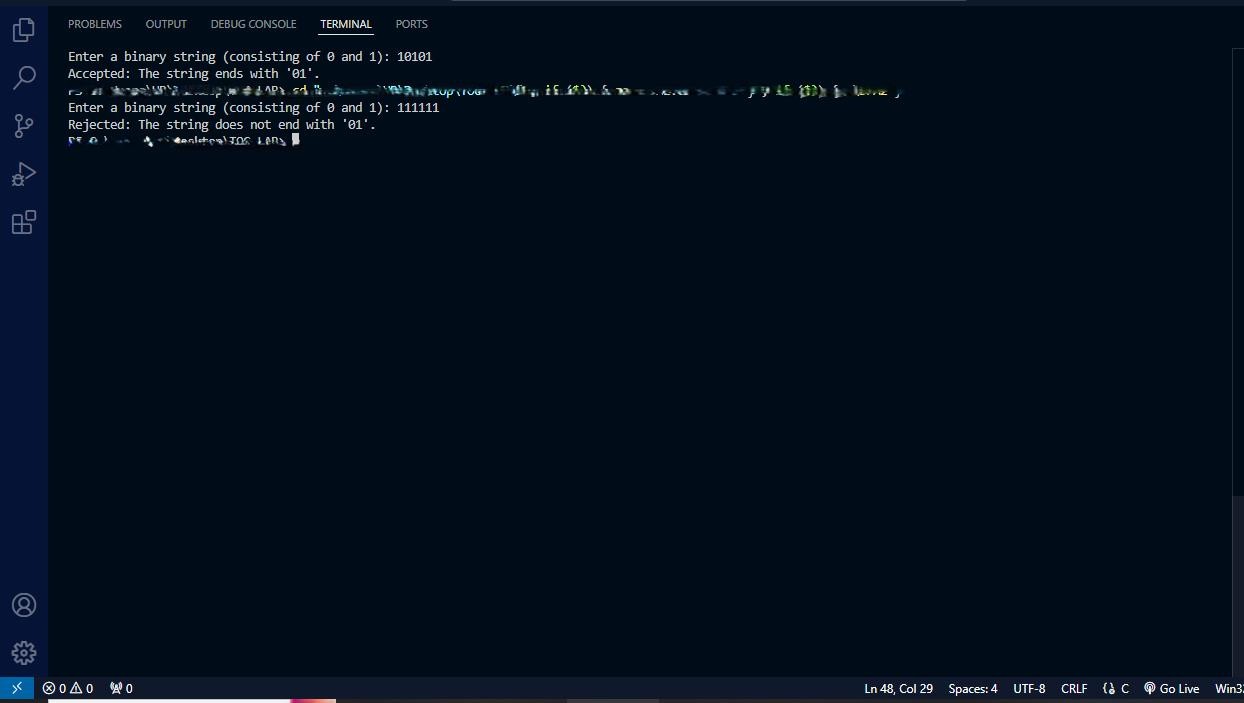
printf("Rejected: The string does not end with '01'.\n");

}

return 0;

}

1. **Output**



1. **Conclusion**

In this lab, we successfully implemented a DFA to check if binary strings end with "01". The program accurately evaluates the input string and provides feedback on its acceptance based on the defined criteria.

**LAB 4: Implementation of an DFA for Strings Containing Substring "001"**

1. **Objective**

To implement a program that checks whether a given binary string contains substring "001" using a DFA.

1. **Theory**

A Deterministic Finite Automaton (DFA) is a theoretical model used to recognize patterns within input strings. A DFA consists of a finite number of states, including an initial state and one or more accepting states. In this lab, we implement a DFA that accepts binary strings composed of 0 and 1, specifically those that have substring sequence "001". The DFA transitions between states based on the input characters, ultimately determining if the string meets the acceptance criteria.

Transition Table:

|  |  |  |
| --- | --- | --- |
| δ | 0 | 1 |
| → q0 | q1 | q0 |
| q1 | q2 | q0 |
| q2 | q2 | qf |
| \*qf | qf | qf |

1. **Source Code**

#include <stdio.h>

enum states {q0,q1,q2,qf};

enum states delta(enum states,char);

int main () {

enum states curr\_state=q0;

char string[20],ch;

int i=0;

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

gets(string);

ch=string [i];

while(ch!='\0')

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

ch=string[++i]; }

if( curr\_state==qf)

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

else

printf("\n The string %s is not valid",string);

return 0; }

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

enum states curr\_state;

switch(s) {

case q0:

if(ch=='0')

curr\_state=q1;

else

curr\_state=q0;

break;

case q1:

if(ch=='0')

curr\_state=q2;

else

curr\_state=q0;

break;

case q2:

if(ch=='0')

curr\_state=q2;

else

curr\_state=qf;

break;

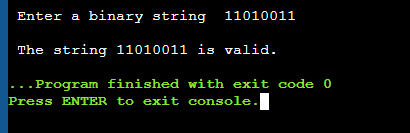
case qf:

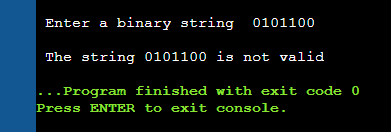
if(ch=='0'|| ch=='1')

curr\_state=qf; }

return curr\_state; }

1. **Output**





1. **Conclusion**

In this lab, we successfully implemented an DFA that checks whether a binary string contains the Substring "001"

**LAB 5: Write a program to validate C identifiers and keywords.**

1. **Objective**

To implement a program to validate C identifiers and keywords

1. **Theory**

* **C identifiers:**

In C programming, **identifiers** are names given to various program elements such as variables, functions, arrays, etc. They are used to uniquely identify these elements within the program. The first character of C identifier must be letter or underscore and remaining characters might be letters, digits or underscore.

* **Keywords:**

**Keywords** are predefined or reserved words that have special meanings to the compiler. These are part of the syntax and cannot be used as identifiers in the programs.All together there are 32 keywords in C. for example: int, char, return, while, void, enum, goto, if, do, float, double

1. **Source code**

#include <stdio.h>

#include<string.h>

char keyword[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"};

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("\nEnter 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; } //end of the main

enum states delta(enum states s,char ch) //Transition function

{ 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; }

return curr\_state; }

int iskeyword(char str[])

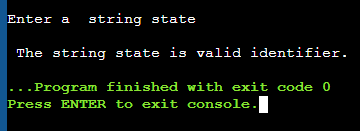
{ for(int i=0;i<32;i++)

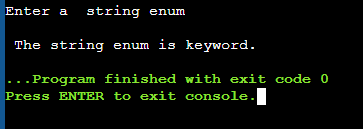
{ if(strcmp(str,keyword[i])==0)

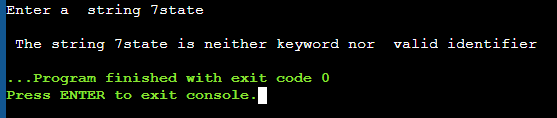
return 1; }

return 0; }

1. **Output**







1. **Conclusion**

In this lab, we successfully implemented an program that validate whether a string is C identifiers and keywords.