**THEORY OF COMPUTATION  
LAB REPORT**



A LAB REPORT SUBMITTED TO

**PATAN MULTIPLE CAMPUS**

**INSTITUTE OF SCIENCE AND TECHNOLOGY**

BY

**SURAB PARAJULI**

**CAMPUS ROLL No: 34**

**SYMBOL No: 80010139**

**T.U. REGISTRATION No: 5-2-22-2430-2023**

**4th sem - ‘A’**

**\_\_\_\_\_\_\_\_\_\_\_\_, 2025**

Table Of Contents

[Common Functions 1](#_Toc207743999)

[1. File handling 2](#_Toc207744000)

[2. Substring-specific 3](#_Toc207744001)

[3. Substring-All 4](#_Toc207744002)

[4. Prefix 5](#_Toc207744003)

[5. Suffix 6](#_Toc207744004)

[6. Substring(s) 7](#_Toc207744005)

[7. DFA-Suffix 9](#_Toc207744006)

[8. DFA-Suffix-2 12](#_Toc207744007)

[9. C Identifiers/Keywords 15](#_Toc207744008)

[10. DFA-Substring 17](#_Toc207744009)

[11. NFA-Prefix 20](#_Toc207744010)

[12. PDA-Equal 0, 1 22](#_Toc207744011)

[13. PDA-NonNull 23](#_Toc207744012)

# Common Functions

Throughout all the programs, there are some common functionalities that are irrelevant to the algorithm/problem that the question is trying to address. For those types of functionalities, to simplify the program, a separate common header file is maintained. The following are the relevant code snippets from the common header file which was compiled and successfully ran with gcc in arch-linux. Full code can be found in GitHub repo (https://github.com/WoodenNebula/lab\_assignment)

Code:

// src/commons.h

#pragma once

#include <stdio.h>

#include <stdlib.h>

#define RED "\033[91m"

#define GREEN "\033[92m"

#define RESET "\033[0m"

void footer() {

  printf(

      "\nSurab Parajuli\nSection: A, 4th-Sem\nRoll: 34, Symbol No.: 80010139");

#ifndef \_\_linux

  // syst("read -p \"Press any key to continue...\"");

  system("pause");

#endif

}

void abortOnError(const char \*msg) {

  printf(RED);

  puts(msg);

  printf(RESET);

  footer();

  exit(EXIT\_FAILURE);

}

# File handling

**Write a C/C++ program to read a file and Count number of lines and characters in a file.**

Code:

#include <filesystem>

#include <fstream>

#include <iostream>

#include "src/commons.h"

int main(int argc, char \*argv[]) {

  std::filesystem::path rp(argv[0]);

  std::filesystem::path fp = std::filesystem::absolute(std::filesystem::relative(rp.parent\_path(), std::filesystem::current\_path()));

  fp.append("01.txt");

  std::ifstream file(fp);

  if (!file) {    std::cerr << "Error: file at " << fp << " couldn't be opened" << std::endl;  }

  int characterCount = 0;

  int lineCount = 0;

  while (true) {

    char c = file.get();

    if (!file.good()) {      break;    }

    if (c == '\n') {      lineCount++;    }

    characterCount++;

  }

  std::cout << "Line = " << lineCount << ", Character(whitespace counted) = " << characterCount << std::endl;

  footer();

}

**Output:**

A computer screen shot of a computer code

AI-generated content may be incorrect.

# Substring-specific

**Write a program to find the substring of a string from specified position up to n length.**Code:

#include <iostream>

#include <string>

#include "src/commons.h"

int main(int argc, char \*argv[]) {

  std::string parentString = "Hello6789World!";

  std::cout << "Parent String = " << parentString << "\n";

  int pos;

  std::cout << "Enter pos in parent string from which substring begins (pos "

               "character exclusive): ";

  std::cin >> pos;

  int count;

  std::cout << "Enter length of substring: ";

  std::cin >> count;

  std::cout << "Substring is: " << parentString.substr(pos, count) << std::endl;

  footer();

  return 0;

}

**Output:**

A computer screen with white text

AI-generated content may be incorrect.

# Substring-All

**Write a program in C/C++ to find all the substrings of a given string.**

Code:

#include <iostream>

#include <string>

#include "src/commons.h"

int main(int argc, char \*argv[]) {

  std::string parentString = "AbcDe";

  std::cout << "Parent String = " << parentString << "\n";

  int startPos = 0;

  while (startPos < parentString.length()) {

    int count = 1;

    while (count <= parentString.length() - startPos) {

      std::cout << parentString.substr(startPos, count) << "\n";

      count++;

    }

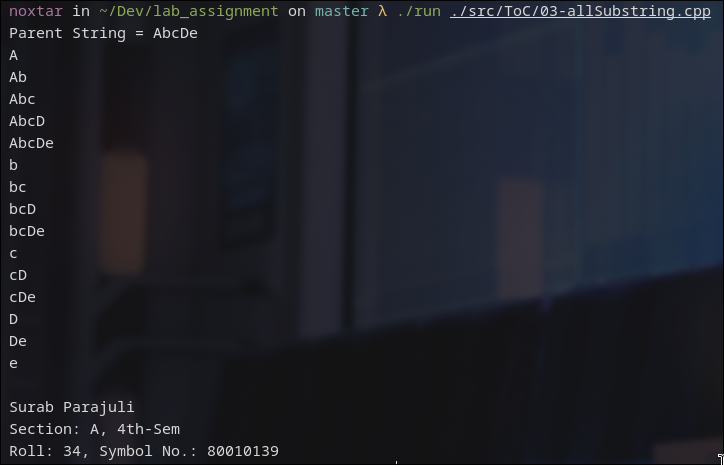
    startPos++;

  }

  footer();

}

**Output:**



# Prefix

**Write C/C++ program for List out the All Prefix of a String.**

Code:

#include <iostream>

#include <string>

#include "src/commons.h"

int main(int argc, char \*argv[]) {

  std::string parentString = "AbcDe";

  std::cout << "Parent String = " << parentString << "\n";

  int count = 1;

  while (count <= parentString.length()) {

    std::cout << parentString.substr(0, count) << "\n";

    count++;

  }

  footer();

}

**Output:**

A computer screen shot of a blue object

AI-generated content may be incorrect.

# Suffix

**Write a C/C++ program for Listing out the All Suffix of a String.**

Code:

#include <algorithm>

#include <iostream>

#include <string>

#include "src/commons.h"

int main(int argc, char \*argv[]) {

  std::string parentString = "AbcDe";

  std::cout << "Parent String = " << parentString << "\n";

  int count = 1;

  for (int pos = parentString.length() - 1; pos >= 0; pos--) {

    std::cout << "count=" << count << ", pos=" << pos << std::endl;

    std::cout << parentString.substr(pos, count) << "\n";

    count++;

  }

  footer();

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

# Substring(s)

**Write a program in C or C++ that accepts a string and print the substrings as below:**

a. The leftmost substring of length n, where n is your input.

b. The rightmost substring of length n, where n is your input.

c. The substring from position n to m symbols where m and n are input

Code:

#include <iostream>

#include <string>

#include "src/commons.h"

void LeftMostSub(const std::string\_view &string) {

  std::cout << "--------\n";

  int length;

  std::cout << "Enter length of left most substring: ";

  std::cin >> length;

  std::cout << "LeftMost substring of length 'n' is: "

            << string.substr(0, length) << "\n";

}

void RightMostSub(const std::string\_view &string) {

  std::cout << "--------\n";

  int length;

  std::cout << "Enter length of right most substring: ";

  std::cin >> length;

  if (length >= 0) {

    int pos = string.length() - length;

    std::cout << "count=" << length << ", pos=" << pos << std::endl;

    std::cout << "LeftMost substring of length 'n' is: "

              << string.substr(pos, length) << "\n";

  } else {

    std::cerr << "[Err] - {Invalid length}" << std::endl;

  }

}

void PositionalSubstring(const std::string\_view &string) {

  std::cout << "--------\n";

  int startPos, endPos;

  std::cout << "For Positional substring, enter:\n\tstartPos(n) = ";

  std::cin >> startPos;

  std::cout << "\tendPos(m) = ";

  std::cin >> endPos;

  if ((startPos >= 0 && endPos >= 0 && startPos <= endPos)) {

    std::cout << "Substring from [n,m]=[" << startPos << ", " << endPos

              << "] is: " << string.substr(startPos, endPos - startPos + 1)

              << "\n";

  } else {

    std::cerr << "[Err] - {Invalid start or end position}" << std::endl;

  }

}

int main(int argc, char \*argv[]) {

  std::cout << "Enter a String = ";

  std::string parentString;

  std ::cin >> parentString;

  LeftMostSub(parentString);

  RightMostSub(parentString);

  PositionalSubstring(parentString);

  footer();

  return 0;

}

**Output:**

A computer screen shot of a computer program

AI-generated content may be incorrect.

# DFA-Suffix

**Construct a DFA accepting strings ending at b, from alphabet {a,b}. Construct its transition diagram and implement your DFA in a program in C or C++.**

DFA:

Code:

#include <iostream>

#include <set>

#include <string>

#include "src/commons.h"

bool isInFinalState = false;

void verdict() {

  std::cout << (isInFinalState ? GREEN : RED);

  std::cout << (isInFinalState ? "Accepted" : "Rejected");

  std::cout << RESET << std::endl;

}

int main(int argc, char \*argv[]) {

  std::cout << "==========\n";

  std::cout << "DFA That accepts strings ending at b from {a,b}\n";

  std::cout << "==========\n";

  std::string string;

  if (argc > 1) {

    string = argv[1];

  } else {

    std::cout << "Enter a string: ";

    std::cin >> string;

  }

  std::cout << "Input:" << string << std::endl;

  std::set<char> alphabet = {'a', 'b'};

  for (char c : string) {

    if (alphabet.find(c) == alphabet.end()) {

      isInFinalState = false;

      break;

    }

    isInFinalState = (c == 'b');

  }

  verdict();

  footer();

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

# DFA-Suffix-2

**Construct transition table for a DFA accepting all strings from {0,1}\* ending with 01. Write a program in C or C++.**

Transition Table:

Code:

#include <iostream>

#include <set>

#include <string>

#include "src/commons.h"

bool isInFinalState = false;

void verdict() {

  std::cout << (isInFinalState ? GREEN : RED);

  std::cout << (isInFinalState ? "Accepted" : "Rejected");

  std::cout << RESET << std::endl;

}

void transition(char c) {

  static bool is0Seen = false;

  static bool is01Seen = false;

  switch (c) {

  case '0':

    is0Seen = true;

    is01Seen = false;

    isInFinalState = false;

    break;

  case '1':

    if (is0Seen) {

      is0Seen = false;

      is01Seen = true;

      isInFinalState = true;

      break;

    } else {

      is0Seen = false;

      is01Seen = false;

      isInFinalState = false;

      break;

    }

  default:

    isInFinalState = false;

    break;

  }

}

int main(int argc, char \*argv[]) {

  std::cout << "==========\n";

  std::cout << "DFA That accepts strings ending at 01 from {0,1}\*\n";

  std::cout << "==========\n";

  std::string string;

  if (argc > 1) {

    string = argv[1];

  } else {

    std::cout << "Enter a string: ";

    std::cin >> string;

  }

  std::cout << "Input:" << string << std::endl;

  std::set<char> alphabet = {'0', '1'};

  // driver code

  for (char c : string) {

    if (alphabet.find(c) == alphabet.end()) {

      isInFinalState = false;

      break;

      std::cerr << RED << "Invalid Char" << RESET << std::endl;

    }

    transition(c);

  }

  verdict();

  footer();

}

**Output:**

A computer screen with white text

AI-generated content may be incorrect.

# C Identifiers/Keywords

**Write a Program to validate C identifiers and keywords.**

Code:

#include <iostream>

#include <set>

#include <string>

#include <string\_view>

#include "src/commons.h"

bool isInFinalState = false;

void verdict() {

  std::cout << (isInFinalState ? GREEN : RED);

  std::cout << (isInFinalState ? "Accepted" : "Rejected");

  std::cout << RESET << std::endl;

}

enum class E\_Alphabet { NONE, DIGIT, LETTER, UNDERSCORE };

E\_Alphabet ClassifySymbol(char c) {

  if (c == '\_') {

    return E\_Alphabet::UNDERSCORE;

  } else if (c >= '0' && c <= '9') {

    return E\_Alphabet::DIGIT;

  } else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {

    return E\_Alphabet::LETTER;

  } else {

    return E\_Alphabet::NONE;

  }

}

bool isValidIdentifier(const std::string\_view &string) {

  if (ClassifySymbol(string[0]) == E\_Alphabet::DIGIT) {

    return false;

  }

  for (char c : string) {

    E\_Alphabet a = ClassifySymbol(c);

    if (a == E\_Alphabet::NONE) {

      return false;

    }

  }

  return true;

}

int main(int argc, char \*argv[]) {

  std::string string;

  if (argc > 1) {    string = argv[1];  }

  std::cout << "Input:" << string << std::endl;

  std::set<std::string\_view> keywords = {

      "const",    "int",      "float",  "double", "long",    "char",

      "void",     "unsigned", "short",  "signed", "size\_t",  "struct",

      "enum",     "union",    "switch", "case",   "default", "if",

      "else",     "while",    "do",     "for",    "break",   "continue",

      "return",   "goto",     "static", "extern", "auto",    "register",

      "volatile", "typedef"};

  isInFinalState = keywords.find(string) != keywords.end();

  if (isInFinalState) {

    verdict();

    std::cout << "--By Keyword" << std::endl;

  } else {

    isInFinalState = isValidIdentifier(string);

    if (isInFinalState) {

      verdict();

      std::cout << "--By Valid Identifier" << std::endl;

    } else {      verdict();    }

  }

  footer();

}

**Output:**

A screenshot of a computer screen

AI-generated content may be incorrect.

# DFA-Substring

**Construct a DFA – Transition graph and Transition table , accepting strings from alphabet {0,1} having substring 010. Write a program in C or C++ to Simulate this DFA.**

DFA:

Code:

#include <iostream>

#include <set>

#include <string>

#include "src/commons.h"

bool isInFinalState = false;

void verdict() {

  std::cout << (isInFinalState ? GREEN : RED);

  std::cout << (isInFinalState ? "Accepted" : "Rejected");

  std::cout << RESET << std::endl;

}

void transition(char c) {

  static bool is0Seen = false;

  static bool is01Seen = false;

  if (isInFinalState) {

    return;

  }

  switch (c) {

  case '0':

    is0Seen = true;

    if (is01Seen) {

      isInFinalState = true;

    }

    break;

  case '1':

    if (is0Seen) {

      is0Seen = false;

      is01Seen = true;

    } else {

      is0Seen = false;

      is01Seen = false;

    }

    break;

  default:

    isInFinalState = false;

    break;

  }

}

int main(int argc, char \*argv[]) {

  std::cout << "==========\n";

  std::cout << "DFA That accepts strings having substring 010 from {0,1}\*\n";

  std::cout << "==========\n";

  std::string string;

  if (argc > 1) {    string = argv[1];  } else {

    std::cout << "Enter a string: ";

    std::cin >> string;

  }

  std::cout << "Input:" << string << std::endl;

  std::set<char> alphabet = {'0', '1'};

  // driver code

  for (char c : string) {

    if (alphabet.find(c) == alphabet.end()) {

      isInFinalState = false;

      break;

      std::cerr << RED << "Invalid Char" << RESET << std::endl;

    }

    transition(c);

  }

  verdict();

  footer();

}

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

# NFA-Prefix

**Design NFA for accepting strings over alphabet {0,1} starting with 01 and write program in C/C++.**

NFA:

Code:

#include <iostream>

#include <set>

#include <string>

#include "src/commons.h"

bool isInFinalState = false;

void verdict() {

  std::cout << (isInFinalState ? GREEN : RED);

  std::cout << (isInFinalState ? "Accepted" : "Rejected");

  std::cout << RESET << std::endl;

}

int main(int argc, char \*argv[]) {

  std::cout << "==========\n";

  std::cout << "NFA That accepts strings starting with 01 from {0,1}\*\n";

  std::cout << "==========\n";

  std::string string;

  if (argc > 1) {    string = argv[1];  } else {

    std::cout << "Enter a string: ";

    std::cin >> string;

  }

  std::cout << "Input:" << string << std::endl;

  std::set<char> alphabet = {'0', '1'};

  isInFinalState = string.substr(0, 2).compare("01") == 0;

  verdict();

  footer();

}

**Output:**

A computer screen shot of a person

AI-generated content may be incorrect.

# PDA-Equal 0, 1

**Write a program to simulate a PDA accepting a language of strings over alphabet {0,1} with equal no of 0s and 1s.**

Code:

#include "src/commons.h"

#include <iostream>

#include <stack>

#include <string\_view>

#include <vector>

bool isInFinalState = false;

void verdict(const std::string\_view &s) {

  std::cout << "Input: " << (s.empty() ? "ε" : s) << "->" << (isInFinalState ? std::string(GREEN) + "Accepted" : std::string(RED) + "Rejected") << RESET << "\n";

}

bool simulatePDA(const std::string\_view &input) {

  std::stack<char> stack;  stack.push('Z'); // bottom marker

  int i = 0;

  for (char c : input) {

    if (c == '0') {      stack.push('X');    }

else if (c == '1') {

      if (stack.empty() || stack.top() != 'X') { return false; }

      stack.pop();

    } else {      return false;    }

  }

  return (stack.size() == 1 && stack.top() == 'Z');

}

int main() {

  std::vector<std::string\_view> testCases = {

      "0011", "000111", "01", "", // valid

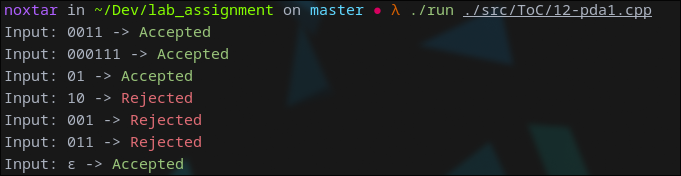
      "10",   "001",    "011" // invalid

  };

  for (auto &s : testCases) {isInFinalState = simulatePDA(s); verdict(s); }

}

**Output:**



# PDA-NonNull

**Construct a PDA accepting language L={0n1n |n>=1 }. Write a program to implement this PDA.**

PDA:

Code:

#include "src/commons.h"

#include <iostream>

#include <stack>

#include <string\_view>

#include <vector>

bool isInFinalState = false;

void verdict(const std::string\_view &s) {

  std::cout << "Input: " << (s.empty() ? "ε" : s) << "->" << (isInFinalState ? std::string(GREEN) + "Accepted" : std::string(RED) + "Rejected") << RESET << "\n";

}

bool simulatePDA(const std::string\_view &input) {

  // dont accept empty string

  if (input.size() == 0) {    return false;  }

  std::stack<char> stack;  stack.push('Z'); // bottom marker

  int i = 0;

  for (char c : input) {

    if (c == '0') {      stack.push('X');    }

else if (c == '1') {

      if (stack.empty() || stack.top() != 'X') { return false; }

      stack.pop();

    } else {      return false;    }

  }

  return (stack.size() == 1 && stack.top() == 'Z');

}

int main() {

  std::vector<std::string\_view> testCases = {

      "0011", "000111", "01", // valid

      "10",   "001",    "011", "" // invalid

  };

  for (auto &s : testCases) {isInFinalState = simulatePDA(s); verdict(s); }

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.