ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ № 3

за курсом “Мовні технології”

студента групи ПЗ-24у-1

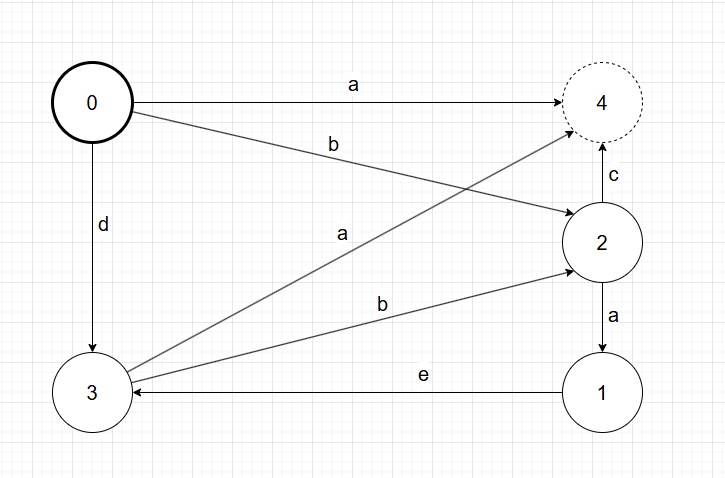
Кондрачука Олександра Вадимовича

2024/2025 н.р.

**1.** **Постановка задачі**

Побудувати на С++ діаграму станів СА, яка розпізнає задані слова.

Діаграма:



Всі регулярні вирази, що описують слова мови цим ДСА: a, bc, baea, b(aeb)\*c, b(aeb)\*a, da, dbc, d(bae)\*a, d(bae)\*bc.

**2. Опис розв’язку**

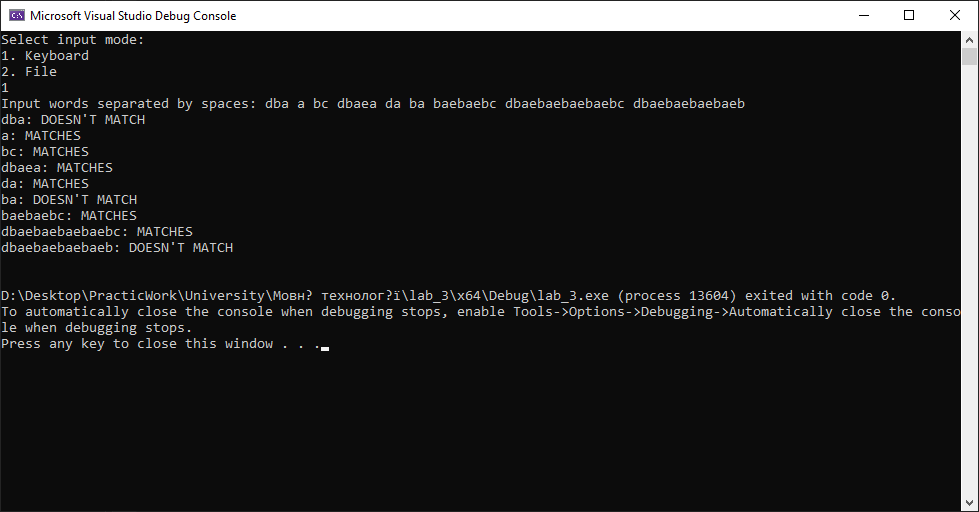
1. Клас FiniteStateMachine:
   * Цей клас є кінцевим автоматом. Він має приватне поле currentState\_, яке зберігає поточний стан автомата.
   * Метод ChangeState змінює стан автомата відповідно до символу.
   * Метод CheckTransitions перевіряє, чи є послідовність символів припустимим словом у цьому автоматі.
2. Основна функція main:
   * Вона відкриває файл з назвою "file\_in.txt" для читання.
   * Кожне слово з файлу додається до вектору words.
   * Для кожного слова викликається метод CheckTransitions для перевірки відповідності слова заданому кінцевому автомату.
   * Результати перевірки записуються в вихідний файл "file\_out.txt".

**3.Опис інтерфейсу**

Створити файл file\_in.txt та заповнити його даними

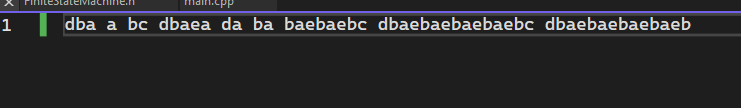
**4. Тестування програми**

**За допомогою клавіатури:**

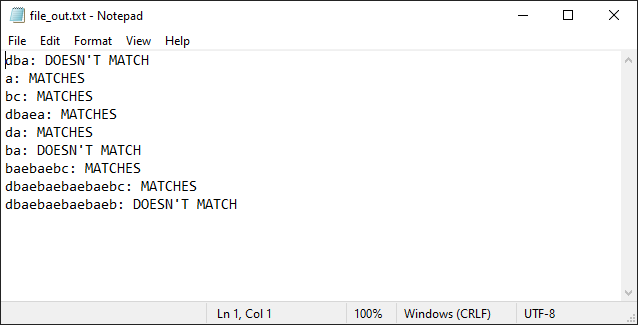
****

**За допомогою файлу:**

**input.txt:**

****

**output.txt:**

****

**5.** **Аналіз помилок**

**немає**

**6. додаток**

#pragma once

#include <map>

#include <string>

#define StatesAmount 5

#define MaxState 4

class FiniteStateMachine

{

private:

std::map<int, std::map<char, int>> states\_;

int currentState\_;

void AddDefaultStates(int statesAmount)

{

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

{

std::map<char, int> transitions;

states\_.insert(std::make\_pair(i, transitions));

}

}

void AddStateTransitions()

{

states\_[0]['a'] = 4;

states\_[0]['b'] = 2;

states\_[0]['d'] = 3;

states\_[1]['e'] = 3;

states\_[2]['a'] = 1;

states\_[2]['c'] = 4;

states\_[3]['a'] = 4;

states\_[3]['b'] = 2;

}

void AddStates()

{

AddDefaultStates(StatesAmount);

AddStateTransitions();

}

public:

FiniteStateMachine()

{

AddStates();

Reset();

}

bool ChangeState(char c)

{

std::map<char, int> transitions = states\_[currentState\_];

std::map<char, int>::iterator transition = transitions.find(c);

if (transition == transitions.end())

{

return false;

}

currentState\_ = transition->second;

return true;

}

void Reset()

{

currentState\_ = 0;

}

bool CheckTransitions(const std::string& transitions)

{

Reset();

for (auto c : transitions)

{

if (!ChangeState(c))

{

return false;

}

}

return currentState\_ == MaxState;

}

};

#include <fstream>

#include <iostream>

#include <sstream>

#include <string>

#include <vector>

#include "FiniteStateMachine.h"

const std::string InputFile = "file\_in.txt";

const std::string OutputFile = "file\_out.txt";

std::vector<std::string> readFromStream(std::stringstream& ss)

{

std::string word;

std::vector<std::string> words;

while (ss >> word)

{

words.push\_back(word);

}

return words;

}

std::vector<std::string> readFromFile()

{

std::ifstream file(InputFile, std::ios::binary);

if (!file.is\_open())

{

std::cout << "Can't read file.\n";

return {};

}

std::stringstream ss;

ss << file.rdbuf();

file.close();

return readFromStream(ss);

}

std::vector<std::string> readFromConsole()

{

std::cin.ignore();

std::cout << "Input words separated by spaces: ";

std::string input;

std::getline(std::cin, input);

std::stringstream ss;

ss << input;

return readFromStream(ss);

}

void output(const std::string& result)

{

std::cout << result << '\n';

std::ofstream outputFile(OutputFile);

if (!outputFile.is\_open())

{

std::cerr << "Couldn't create the output file" << '\n';

return;

}

outputFile << result;

outputFile.close();

}

int main()

{

std::cout << "Select input mode:\n1. Keyboard\n2. File" << '\n';

std::vector<std::string> words;

int path;

std::cin >> path;

if (path == 1)

{

words = readFromConsole();

}

else

{

words = readFromFile();

}

FiniteStateMachine stateMachine;

std::string result;

for (const std::string& word : words)

{

result += word + ": " + (stateMachine.CheckTransitions(word) ? "MATCHES" : "DOESN'T MATCH") + "\n";

}

output(result);

return 0;

}