# LAPORAN PRAKTIKUM
# KONSTRUKSI PERANGKAT LUNAK

## MODUL IV

## AUTOMATA DAN TABLE-DRIVEN CONSTRUCTION


**Disusun Oleh :**

**Dhiya Ulhaq Ramadhan**

**S1SE-06-02**

**Asisten Praktikum :**

**Muhamad Taufiq Hidayat**

**Dosen Pengampu :**

**Riyan Dwi Yulian Prakoso, S.Kom., M.Kom.**



**PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK**

**DIREKTORAT TELKOM KAMPUS PURWOKERTO**

**2025**

# BAB I
# PENDAHULUAN

## A. MAKSUD DAN TUJUAN

1. Menguasai penerapan automata-based construction pada Java Script
2. Menguasai penerapan table-based construction pada Java Script

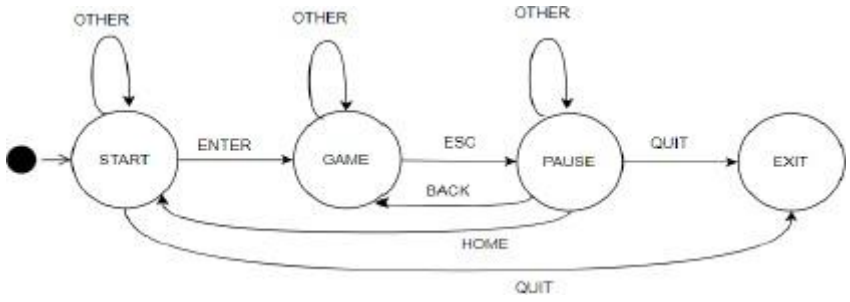
## B. DASAR TEORI

1. Automata Construction

Automata Construction atau Automata-based programming adalah salah satu paradigma pemrograman dimana program dianggap seperti finite-state machine (FMS) atau formal automaton lainnya yang memiliki berbagai state yang saling berkaitan dan memiliki aturan tertentu yang jelas. Berikut ini indikator utama dalam Automata-based programming:

a. Jangka waktu eksekusi program dipisahkan dengan jelas pada state yang ada dan tidak terjadi eksekusi yang overlaping pada state state yang ada.
b. Semua komunikasi antara state state yang ada (perpindahan antar state) hanya dapat dilakukan secara eksplisit yang disimpan pada suatu global variable.

1.1. Implementasi




```
using System;

public class Program
{
    enum State { START, GAME, PAUSE, EXIT }

    public static void Main()
    {
        State state = State.START;
        string[] screenName = { "START", "GAME", "PAUSE", "EXIT"
};

        while (state != State.EXIT)
        {
            Console.WriteLine(screenName[(int)state] + "
SCREEN");
            Console.Write("Enter Command: ");
            string command = Console.ReadLine();

            switch (state)
            {
```

```
                        case State.START:
                            if (command == "ENTER")
                                state = State.GAME;
                            else if (command == "QUIT")
                                state = State.EXIT;
                            break;

                        case State.GAME:
                            if (command == "ESC")
                                state = State.PAUSE;
                            break;

                        case State.PAUSE:
                            if (command == "BACK")
                                state = State.GAME;
                            else if (command == "HOME")
                                state = State.START;
                            else if (command == "QUIT")
                                state = State.EXIT;
                            break;
                    }
                }

                Console.WriteLine("EXIT SCREEN");
            }
}
```

1.2. Table-driven Construction

Table-driven Construction adalah skema yang memunngkinkan mencari informasi menggunakan table dibandingkan menggunakan logic statement (if dan case) untuk mengetahuinya. Hampir semua hal yang dapat ditangani oleh if dan case, dapat diubah menjadi table-driven sebagai gantinya. Dalam kasus sederhana, pernyataan logika lebih mudah dan lebih langsung. Saat logic statements sudah menjadi begitu kompleks, penggunaan teable-driven akan menjadi semakin menarik.

Table-driven memiliki dua hal yang perlu diperhatikan sebelum diimplementasikan. Pertama kita perlu menentukan bagaimana mencari entri pada table. Kedua beberapa tipe data tidak bisa digunakan untuk mencari entri pada table secara langsung. Secara umum cara pencarian entri pada table-driven dibagi menjadi tiga yaitu :

1. Direct Access
2. Indexed Access
3. Stair-step Access

1.2.1. Direct Access

Secara sederhana direct access adalah metode table-driven yang Secara langsung mengubah kondisi pada logic statement menjadi key dari table yang digunakan. Untuk lebih jelasnya perhatikan potongan kode berikut.

```
public static int GetDaysPerMonth(int month){
      int[] daysPerMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30,
31, 30, 31 };
      return daysPerMonth[month - 1];
}
```

1.2.2. Indexed Access

Indexed Acces adalah metode table-driven yang mengunakan table lain untuk menyimpan index dari table utama, hal ini bertujuan untuk mengurangi penggunan memori penyimpanan dari program apabila ukuran satu dari entri table utama besar dan memiliki nilai entri yang berulang.

### 1.2.3. Stair-step Access

Salah satu metode dalam table-driven yang berguna untuk mempermudah pencarian entri apabila terdapat kasus dimana nilai yang didapatkan berdasarkan range yang telah ada seperti contoh index nilai mahasiswa. Untuk lebih jelasnya perhatikan potongan kode berikut.

```
public static string GetGradeByScore(double studentScore) {
            string[] grade = { "A", "AB", "B", "BC", "C", "D", "E"
        };
            double[] rangeLimit = { 80.0, 70.0, 65.0, 60.0, 50.0,
        40.0, 0.0 };
            int maxGradeLevel = grade.Length - 1;

            string studentGrade = "E";
            int gradeLevel = 0;
            while  ((studentGrade  ==  "E")  &&  (gradeLevel  <
        maxGradeLevel))
            {
                if (studentScore > rangeLimit[gradeLevel])
                    studentGrade = grade[gradeLevel];

                gradeLevel++;
            }

            return studentGrade;
}
```

# BAB II
# IMPLEMENTASI (GUIDED)

1. Automata-based Construction

   Input :

```
        const readline = require("readline");

        const rl = readline.createInterface({
            input: process.stdin,
            output: process.stdout
        });

        const State = {
            START: "START",
            GAME: "GAME",
            PAUSE: "PAUSE",
            EXIT: "EXIT"
        };

        let state = State.START;

        function runStateMachine() {
            console.log(`${state} SCREEN`);
            rl.question("Enter Command: ", (command) => {
                switch (state) {
                    case State.START:
                        if (command === "ENTER") state = State.GAME;
                        else if (command === "QUIT") state = State.EXIT;
                        break;
                    case State.GAME:
                        if (command === "ESC") state = State.PAUSE;
                        break;
                    case State.PAUSE:
                        if (command === "BACK") state = State.GAME;
                        else if (command === "HOME") state = State.START;
                        else if (command === "QUIT") state = State.EXIT;
                        break;
                }
                if (state !== State.EXIT) {
                    runStateMachine();
                } else {
                    console.log("EXIT SCREEN");
                    rl.close();
                }
            });
        }

runStateMachine();
```

Output :

```
PS D:\bersama berkarya\SEMESTER 6\Praktikum KPL\KPL_Dhiya_Ulhaq_Ramadhan_2211104053_SE0602> node "Praktikum_2\guided.js"
START SCREEN
Enter Command: SCREEN
START SCREEN
Enter Command: GAME
START SCREEN
Enter Command: PAUSE
START SCREEN
Enter Command: EXIT
START SCREEN
Enter Command: QUIT
EXIT SCREEN
PS D:\bersama berkarya\SEMESTER 6\Praktikum KPL\KPL_Dhiya_Ulhaq_Ramadhan_2211104053_SE0602>
```

2. Table-driven Construction

2.1. Direct Access

Input :

```
function getDaysPerMonth(month) {
     const daysPerMonth = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];
     return daysPerMonth[month - 1] || "Invalid month";
}

console.log(getDaysPerMonth(2)); // Output: 28
console.log(getDaysPerMonth(13)); // Output: Invalid month
```

Output :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
C:\Program Files\nodejs\node.exe .\Praktikum_2\direct.js
28
```

2.2. Stair-step Access

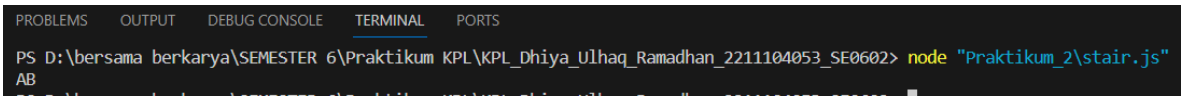
Input :

```
function getGradeByScore(studentScore) {
    const grades = ["A", "AB", "B", "BC", "C", "D", "E"];
    const rangeLimit = [80, 70, 65, 60, 50, 40, 0];

    for (let i = 0; i < rangeLimit.length; i++) {
        if (studentScore >= rangeLimit[i]) {
            return grades[i];
        }
     }
     return "E";
}

console.log(getGradeByScore(75)); // Output: AB
console.log(getGradeByScore(45)); // Output: D
```

Output :


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS D:\bersama berkarya\SEMESTER 6\Praktikum KPL\KPL_Dhiya_Ulhaq_Ramadhan_2211104053_SE0602> node "Praktikum_2\stair.js"
AB

## BAB III
## PENUGASAN (UNGUIDED)

1. Soal 1: Automata-based Construction (FSM)
   Sebuah game memiliki tiga state utama:
   - **START** (awal permainan)
   - **PLAYING** (sedang bermain)
   - **GAME OVER** (permainan berakhir)

   Aturan transisi antar state:
   1. Dari **START**, jika pemain mengetik **"PLAY"**, permainan masuk ke state **PLAYING**.
   2. Dari **PLAYING**, jika pemain mengetik **"LOSE"**, permainan masuk ke state **GAME OVER**.
   3. Dari **GAME OVER**, jika pemain mengetik **"RESTART"**, permainan kembali ke state **START**.
   4. Pemain bisa keluar kapan saja dengan mengetik **"EXIT"**.

JAWABAN DI PAGE SELANJUTNYA DIBAWAH

```
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

const State = {
  START: "START",
  PLAYING: "PLAYING",
  GAME_OVER: "GAME_OVER",
  EXIT: "EXIT"
};

let state = State.START;

function runStateMachine() {
  console.log(`Current state: ${state}`);

  rl.question("Enter command: ", (command) => {
    switch (state) {
      case State.START:
        if (command === "PLAY") {
          console.log("Game is starting...");
          state = State.PLAYING;
        } else if (command === "EXIT") {
          state = State.EXIT;
        } else {
          console.log("Invalid command for START state. Try 'PLAY' or 'EXIT'");
        }
        break;

      case State.PLAYING:
        if (command === "LOSE") {
          console.log("You lost the game!");
          state = State.GAME_OVER;
        } else if (command === "EXIT") {
          state = State.EXIT;
        } else {
          console.log("Invalid command for PLAYING state. Try 'LOSE' or 'EXIT'");
        }
        break;

      case State.GAME_OVER:
        if (command === "RESTART") {
          console.log("Restarting the game...");
          state = State.START;
        } else if (command === "EXIT") {
          state = State.EXIT;
        } else {
          console.log("Invalid command for GAME_OVER state. Try 'RESTART' or
'EXIT'");
        }
        break;
    }

    if (state === State.EXIT) {
      console.log("Exiting game. Goodbye!");
      rl.close();
    } else {
      runStateMachine();
    }
  });
}

console.log("Welcome to the Game State Machine!");
console.log("Commands: 'PLAY', 'LOSE', 'RESTART', 'EXIT'");
runStateMachine();
```

Output :

```
PS D:\bersama berkarya\SEMESTER 6\Praktikum KPL\KPL_Dhiya_Ulhaq_Ramadhan_2211104053_SE0602\Praktikum_2> node unguided.js
Welcome to the Game State Machine!
Commands: 'PLAY', 'LOSE', 'RESTART', 'EXIT'
Current state: START
Enter command: PLAY
Game is starting...
Current state: PLAYING
Enter command: LOSE
You lost the game!
Current state: GAME_OVER
Enter command: RESTART
Restarting the game...
Current state: START
Enter command: EXIT
Exiting game. Goodbye!
```

Penjelasan Kode :

Program dimulai dengan mengimpor modul readline yang memungkinkan interaksi dengan pengguna melalui terminal. Kemudian interface readline dibuat untuk mengambil input dari pengguna dan memberikan output ke konsol.

State dalam game didefinisikan dalam objek State yang berisi empat kemungkinan state:

a. START: Kondisi awal permainan

b. PLAYING: Kondisi ketika pemain sedang bermain

c. GAME_OVER: Kondisi ketika permainan berakhir

d. EXIT: Kondisi khusus untuk keluar dari permainan

Program ini bersifat rekursif, di mana runStateMachine() akan terus memanggil dirinya sendiri sampai state berubah menjadi EXIT. Ketika mencapai state EXIT, program akan mengucapkan selamat tinggal dan menutup interface readline, yang mengakhiri program.