LAPORAN PRAKTIKUM
KONSTRUKSI PERANGKAT BERGERAK

MODUL II

AUTOMATA DAN TABLE-DRIVEN CONSTRUCTION

Disusun Oleh :

Aditya Prabu Mukti

2211104037

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. DASAR TEORI

1.Automata Construction

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

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

2. Table-driven Construction

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

Table-driven memiliki dua hal yang perlu diperhatikan sebelum diimplementasikan. Pertama kita perlu menentukan bagaimana mencari entri pada tabel. 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.

B. MAKSUD DAN TUJUAN

1. Menguasai penerapan automata-based construction pada C#.Net
2. Menguasai penerapan table-based construction pada C#.Net

BAB II
IMPLEMENTASI (GUIDED)

1.Automata-based Construction

```
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:\SEMESTER 6\KONSTRUKSI PERANGKAT LUNAK\praktikum\pertemuan2> node app.js
START SCREEN
Enter Command: ENTER
GAME SCREEN
Enter Command: QUIT
GAME SCREEN
Enter Command: ESC
PAUSE SCREEN
Enter Command: HOME
START SCREEN
Enter Command: QUIT
EXIT SCREEN
PS D:\SEMESTER 6\KONSTRUKSI PERANGKAT LUNAK\praktikum\pertemuan2>
```

Penjelasan:

Program dimulai dalam state START, lalu pengguna dapat memasukkan perintah tertentu untuk berpindah ke state lain. Jika pengguna mengetik "ENTER", maka state akan berpindah ke GAME, yang merepresentasikan bahwa permainan telah dimulai. Jika dalam state GAME pengguna mengetik "ESC", maka permainan akan dijeda dan masuk ke PAUSE. Dari state PAUSE, pengguna bisa mengetik "BACK" untuk kembali ke GAME, "HOME" untuk kembali ke START, atau "QUIT" untuk keluar dari aplikasi dengan berpindah ke EXIT.

2.Table-Driven Construction

Mounth.js

```
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:

```
PS D:\SEMESTER 6\KONSTRUKSI PERANGKAT LUNAK\praktikum\pertemuan2> node mounth.js
28
Invalid month
PS D:\SEMESTER 6\KONSTRUKSI PERANGKAT LUNAK\praktikum\pertemuan2>
```

Stair-StepAccess.js

```
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(0));
console.log(getGradeByScore(60));
```

Output:

```
PS D:\SEMESTER 6\KONSTRUKSI PERANGKAT LUNAK\praktikum\pertemuan2> node Stair-StepAccess.js
E
BC
PS D:\SEMESTER 6\KONSTRUKSI PERANGKAT LUNAK\praktikum\pertemuan2>
```

Penjelasan:

Pada Direct Access, kode menggunakan array daysPerMonth untuk menyimpan jumlah hari dalam setiap bulan. Dengan pendekatan ini, program dapat langsung mengakses jumlah hari berdasarkan indeks bulan yang diberikan, tanpa perlu menggunakan banyak pernyataan if-else. Misalnya, ketika pengguna memanggil getDaysPerMonth(2), program langsung mengambil elemen kedua dari array, yaitu 28, yang menunjukkan jumlah hari dalam bulan Februari.

Sementara itu, pada Stair-step Access, program menggunakan dua array, yaitu grades yang berisi daftar nilai huruf dan rangeLimit yang berisi batas nilai numerik. Ketika pengguna memasukkan skor tertentu, program akan melakukan iterasi terhadap array rangeLimit untuk menemukan rentang nilai yang sesuai, lalu mengembalikan huruf yang sesuai dari array grades. Teknik ini sering digunakan dalam sistem penilaian akademik, pemetaan data, serta pengambilan keputusan berbasis aturan.

BAB III
PENUGASAN (UNGUIDED)

Game_fsm.js

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

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

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

let state = State.START;

function runStateMachine() {
    console.log(`Current State: ${state}`);
    rl.question("Enter Command: ", (command) => {
        switch (state) {
            case State.START:
                if (command === "PLAY") state = State.PLAYING;
                else if (command === "EXIT") return exitGame();
                break;
            case State.PLAYING:
                if (command === "LOSE") state = State.GAME_OVER;
                else if (command === "EXIT") return exitGame();
                break;
            case State.GAME_OVER:
                if (command === "RESTART") state = State.START;
                else if (command === "EXIT") return exitGame();
                break;
        }
        runStateMachine();
    });
}

function exitGame() {
    console.log("Exiting Game...");
    rl.close();
}

runStateMachine();
```

Penjelasan:

Program game_fsm.js adalah implementasi dari Finite State Machine (FSM) untuk sebuah game sederhana dengan tiga state utama: START, PLAYING, dan GAME OVER. Program ini menggunakan readline untuk menerima input pengguna melalui terminal dan mengontrol transisi antar state berdasarkan perintah yang dimasukkan. Awalnya, permainan berada dalam state START, dan jika pemain mengetik "PLAY", permainan akan berpindah ke state PLAYING, yang menunjukkan bahwa permainan sedang berlangsung. Selama di state PLAYING, jika pemain mengetik "LOSE", maka permainan akan berakhir dan masuk ke state GAME OVER. Di state ini, pemain bisa memilih untuk mengetik "RESTART" agar permainan kembali ke state START, atau keluar dari game kapan saja dengan mengetik "EXIT".

Fungsi utama dalam kode ini adalah runStateMachine(), yang mencetak state saat ini dan meminta input pengguna untuk menentukan transisi ke state berikutnya. Setiap kali ada perubahan state, fungsi ini dipanggil kembali untuk terus menjalankan loop hingga pemain mengetik "EXIT", yang akan memanggil fungsi exitGame() untuk menutup antarmuka readline dan menghentikan program. Dengan pendekatan FSM ini, program dapat mengelola alur permainan dengan lebih terstruktur dan fleksibel.

Output: