Nama : Fadhil Dzikri Aqila
NIM : 1103213136
Kelas : TK-45-G09

Analisis Learn Rust with Me! Part 1-6

1. **Hello-cargo**

   Kode ini adalah contoh sederhana dari program Rust untuk mencetak "Hello, world!" menggunakan fungsi main sebagai titik awal eksekusi. Macro println! digunakan untuk mencetak teks ke konsol, menunjukkan sintaks bawaan Rust yang mendukung pencetakan secara langsung. Program ini mencerminkan struktur dasar aplikasi Rust yang dibuat menggunakan Cargo, alat pengelola proyek dan paket untuk Rust. Kode ini memperlihatkan bagaimana Rust dirancang untuk memulai pengembangan dengan cara yang sederhana namun kuat, sesuai prinsip *default setup*.

2. **Data-types**

   Kode ini menunjukkan berbagai cara Rust menangani tipe data melalui penggunaan struct, tuple struct, variabel immutable dan mutable, serta berbagai tipe data dasar seperti integer, float, boolean, dan string. Struct klasik memungkinkan pengelolaan data dengan field bernama, sedangkan tuple struct hanya mendefinisikan tipe data tanpa nama field. Rust mendukung shadowing untuk memperbarui nilai variabel secara aman tanpa menimpa deklarasi awal. Dengan tipe data statis, Rust memastikan kejelasan dan keamanan tipe selama kompilasi. Kode ini menampilkan kemampuan Rust untuk menangani data secara fleksibel, baik dalam bentuk primitif maupun struktur kompleks.

3. **If-else**

   Kode ini memperlihatkan penggunaan pernyataan if-else dalam Rust yang tidak hanya memungkinkan evaluasi kondisi, tetapi juga dapat mengembalikan nilai. Ekspresivitas ini mempermudah penetapan nilai langsung ke variabel berdasarkan kondisi tertentu. Evaluasi kondisi berlapis menggunakan if-else if-else mencerminkan bagaimana Rust mendorong penulisan kode yang eksplisit dan terstruktur. Selain itu, Rust memaksa inisialisasi variabel sebelum digunakan, memastikan bahwa kode bebas dari akses nilai yang tidak terdefinisi. Fitur ini mencerminkan pendekatan Rust yang mengutamakan keamanan dan keandalan kode.

4. **Arrays-and-vectors**

   Kode ini menjelaskan cara Rust menangani koleksi data menggunakan array dan vektor. Array adalah kumpulan dengan ukuran tetap dan tipe data yang sama, sementara vektor menawarkan fleksibilitas untuk menambah atau menghapus elemen. Operasi seperti push dan pop pada vektor menunjukkan bagaimana Rust memungkinkan manipulasi data secara dinamis. Vektor

dalam Rust memerlukan deklarasi mut jika akan diubah, mencerminkan filosofi keamanan Rust dengan membedakan data yang mutable dan immutable. Dengan pendekatan ini, Rust mempermudah pengelolaan koleksi data secara aman dan efisien.

5. Loops

Kode ini menampilkan berbagai bentuk loop di Rust, termasuk loop tanpa henti, while loop, dan for loop. Loop tanpa henti dapat dihentikan menggunakan break, bahkan dengan mengembalikan nilai pada titik tersebut. While loop memungkinkan iterasi selama kondisi tertentu terpenuhi, sedangkan for loop menggunakan iterator untuk memproses koleksi atau rentang nilai. Dengan kombinasi ini, Rust menyediakan fleksibilitas tinggi untuk pengulangan, mendukung pengembang untuk menangani berbagai skenario iterasi dengan sintaks yang sederhana namun jelas.

6. Hash-map

Kode ini menyoroti penggunaan HashMap dalam Rust, koleksi kunci-nilai yang disediakan melalui modul std::collections. Operasi seperti insert untuk menambah pasangan kunci-nilai, get untuk mengambil nilai berdasarkan kunci, dan remove untuk menghapus kunci menunjukkan kapabilitas dasar HashMap. Nilai yang diambil dengan get berupa Option, mencerminkan sistem tipe Rust yang aman untuk menangani kemungkinan ketidakadaan nilai. Dengan mendesain operasi hash map secara eksplisit dan aman, Rust memungkinkan pengelolaan data asosiatif dengan efisien sambil mencegah bug yang umum terjadi pada koleksi kunci-nilai.

Analisis Project Rust

1. Perencanaan Jalur Sederhana

Kode ini menggunakan algoritma *Breadth-First Search* (BFS) untuk menemukan jalur terpendek dari titik awal ke titik tujuan dalam grid 2D. Grid diwakili oleh matriks di mana nilai 0 menunjukkan jalan yang bisa dilalui dan 1 adalah rintangan. Fungsi is_valid_move memastikan pergerakan valid dengan memeriksa batas grid, kondisi jalan, dan status kunjungan. BFS diimplementasikan menggunakan VecDeque untuk mengelola antrian posisi yang perlu dieksplorasi. Jalur direpresentasikan sebagai urutan koordinat (x, y), yang diperbarui pada setiap langkah jika kondisi valid terpenuhi. Program ini menunjukkan pendekatan yang efisien dan aman untuk pencarian jalur dengan memastikan tidak ada posisi yang dilalui lebih dari sekali.

2. Gerakan Robot dengan Input Pengguna

Kode ini mensimulasikan gerakan robot di grid 2D berdasarkan input pengguna. Posisi robot diperbarui sesuai dengan perintah yang dimasukkan (up, down, left, atau right). Validasi perintah dilakukan menggunakan pola match, di mana perintah yang tidak dikenal akan menghasilkan pesan

kesalahan. Perintah exit digunakan untuk keluar dari program. Dengan loop tak terbatas, kode ini terus menerima input dan mencetak posisi robot secara dinamis. Pendekatan ini efektif untuk simulasi robot berbasis perintah langsung, memungkinkan pengguna untuk mengontrol gerakan secara real-time.

3. Simulasi Robot Menghindari Rintangan

Kode ini memperluas contoh perencanaan jalur dengan memvisualisasikan jalur robot dalam grid dengan rintangan. Sama seperti pada kode perencanaan jalur sederhana, algoritma BFS digunakan untuk mencari jalur dari titik awal ke tujuan. Namun, kode ini menambahkan elemen visualisasi grid dan langkah-langkah perjalanan, memperlihatkan bagaimana robot menghindari rintangan dan melaporkan jalur yang diambil. Dengan mencetak setiap langkah perjalanan, kode ini memberikan wawasan yang lebih mendalam tentang proses eksplorasi robot di lingkungan yang kompleks.

4. Penjadwalan Robot dengan Prioritas

Kode ini mensimulasikan penjadwalan tugas robot menggunakan struktur data *BinaryHeap*. Tugas diberikan prioritas, di mana tugas dengan prioritas tertinggi dieksekusi lebih dulu. Prioritas diimplementasikan dengan membalik urutan *heap* menggunakan metode cmp pada struktur Task. Saat tugas ditambahkan ke antrean, tugas-tugas dieksekusi dalam urutan prioritas dengan menggunakan operasi pop. Pendekatan ini sangat efisien untuk manajemen tugas dinamis, memastikan bahwa tugas-tugas penting dieksekusi lebih awal, yang merupakan kebutuhan umum dalam sistem robotik.

5. Robotik dengan Sistem Event-Driven

Kode ini memanfaatkan paradigma *event-driven* untuk mengontrol perilaku robot dalam merespons berbagai kejadian, seperti mendeteksi rintangan atau perubahan target. Robot memiliki posisi dan target, serta dapat bergerak menuju target atau berhenti ketika menghadapi rintangan. Event disimulasikan menggunakan thread terpisah yang secara periodik menghasilkan kejadian-kejadian seperti ObstacleDetected, TargetChanged, atau Idle. Sinkronisasi dilakukan menggunakan Mutex untuk memastikan akses aman ke data bersama. Pendekatan ini meniru sistem robotik dunia nyata yang bereaksi terhadap lingkungan secara asinkron dan responsif.

6. Robot dengan Model Probabilistik

Kode ini menggunakan pendekatan *particle filter* untuk memperkirakan posisi robot dalam lingkungan dengan ketidakpastian. Setiap partikel merepresentasikan kemungkinan posisi robot, dengan posisi dan orientasi diupdate berdasarkan pergerakan dan sensor. Partikel diberi bobot berdasarkan kesesuaian antara data sensor dan prediksi model lingkungan. Setelah pembaruan bobot, resampling dilakukan untuk memilih partikel-partikel yang lebih sesuai. Proses ini

meningkatkan akurasi estimasi posisi robot seiring waktu. Dengan simulasi data sensor dan peta sederhana, kode ini menunjukkan bagaimana algoritma probabilistik dapat digunakan untuk navigasi robotik dalam lingkungan yang tidak pasti.