# Analisis

## Analisis Learn Rust with Me!

- Hello World!

  Rust adalah bahasa pemrograman yang dikenal dengan fokus pada keamanan memori, performa tinggi, dan pengelolaan paralelisme tanpa risiko data race. Proses instalasi Rust dapat dilakukan dengan menggunakan rustup, sebuah toolchain manager yang memungkinkan pengguna untuk mengelola berbagai versi Rust dan alat-alat terkait. Setelah instalasi selesai, kode Rust dapat dijalankan secara manual dengan mengompilasi file .rs menggunakan rustc dan menjalankan hasil kompilasi tersebut sebagai file biner. Namun, penggunaan Cargo, yang merupakan package manager dan build system bawaan Rust, jauh lebih efisien untuk proyek yang lebih kompleks. Dengan Cargo, pengguna dapat dengan mudah membuat proyek baru, mengelola dependensi, dan menjalankan kode hanya dengan perintah sederhana seperti cargo new, cargo build, dan cargo run. Cargo tidak hanya menyederhanakan proses build, tetapi juga menangani pengelolaan paket dan versi secara otomatis, menjadikannya pilihan yang lebih disarankan untuk mengembangkan aplikasi Rust.

- Data Type
  Dalam contoh program ini, saya mempelajari beberapa konsep dasar di Rust, termasuk struktur klasik dan struktur tuple, serta cara mendeklarasikan dan menggunakan variabel dalam Rust. Program dimulai dengan mendefinisikan sebuah struktur Student yang memiliki tiga field: name (String), level (u8), dan remote (bool). Selain itu, ada struktur tuple Grades, yang hanya menyimpan tipe data yang berbeda di dalamnya, seperti char dan f32.

  Di dalam fungsi main(), program menunjukkan cara kerja variabel di Rust dengan mengikat nilai yang tidak dapat diubah (immutable) dan kemudian menggunakan kata kunci mut untuk membuat variabel yang dapat diubah (mutable). Program juga menunjukkan konsep variable shadowing, di mana nilai variabel birth_year dimodifikasi, tetapi nama variabel yang sama digunakan lagi dalam ruang lingkup yang berbeda. Selain itu, Rust memperkenalkan berbagai tipe data seperti f32 untuk angka desimal, bool untuk nilai benar/salah, dan char serta String untuk karakter dan string.

  Program ini juga memperkenalkan tuple sebagai cara untuk mengelompokkan beberapa nilai dengan tipe yang berbeda dalam satu struktur yang tidak dapat diubah ukurannya. Selain itu, menggunakan struktur klasik memungkinkan kita untuk membuat objek dengan berbagai atribut, dan kita bisa mengaksesnya menggunakan nama field yang telah ditentukan. Struktur tuple dapat diakses dengan menggunakan indeks posisi elemen. Secara keseluruhan, program ini memberikan gambaran umum yang baik tentang bagaimana variabel, tipe data, dan struktur bekerja di Rust.
- If-else Statement
  Saya telah mempelajari Rust dan Cargo untuk menjalankan program, dan sekarang saya mencoba membuat program dengan menggunakan pernyataan if/else. Dalam contoh ini, saya memulai dengan perbandingan sederhana untuk memeriksa apakah dua angka sama atau tidak. Selanjutnya, saya menggunakan if/else untuk mengikat nilai

ke variabel take_jacket berdasarkan kondisi cuaca, di mana jika hari cerah, saya akan memutuskan untuk tidak membawa jaket. Terakhir, saya mengevaluasi beberapa kondisi untuk memeriksa apakah suatu angka berada di luar rentang yang diinginkan, dengan menggunakan beberapa pernyataan if/else berturut-turut. Program ini menunjukkan bagaimana cara kerja pengkondisian dan penetapan nilai dalam Rust.

- Arrays and Vectors
  Dalam contoh program ini, saya mempelajari penggunaan arrays dan vectors di Rust, yang keduanya digunakan untuk menyimpan kumpulan nilai dengan tipe data yang sama.

  Arrays adalah koleksi yang berisi elemen-elemen dari tipe data yang sama, dan ukurannya tetap setelah deklarasi. Di sini, saya mendeklarasikan array working_days yang berisi nama-nama hari kerja dan working_days_num yang berisi angka dengan nilai default. Ukuran array dapat ditentukan langsung oleh compiler atau secara eksplisit, seperti pada contoh working_days_num = [0; 5], yang menginisialisasi array berisi lima elemen dengan nilai 0. Saya juga menunjukkan bagaimana mengakses elemen array menggunakan indeks.

  Di sisi lain, vectors adalah koleksi dinamis yang juga menyimpan nilai dari tipe data yang sama, namun ukurannya bisa berubah seiring waktu. Dengan vec![], saya dapat mendeklarasikan dan menginisialisasi vector seperti pada nephews_age. Vector lebih fleksibel dibanding array karena dapat ditambah atau dikurangi elemennya dengan menggunakan metode seperti push() dan pop(). Program ini juga menunjukkan bagaimana cara mengakses elemen berdasarkan posisi dan memodifikasi nilai elemen dalam vector, seperti mengganti nama buah dalam vector fruit.
- Loops
  Dalam contoh program ini, saya mempelajari penggunaan berbagai jenis loop di Rust, termasuk loop tak terbatas, while loop, dan for loop.

  Loop tak terbatas (infinite loop) dimulai dengan kata kunci loop, yang akan terus menjalankan blok kode tanpa henti, kecuali jika ada pernyataan break yang menghentikan eksekusi. Pada contoh ini, loop tersebut tidak dieksekusi karena saya menonaktifkannya dengan komentar, namun biasanya digunakan ketika ingin menjalankan sesuatu berulang kali tanpa batasan. Dengan menambahkan kondisi break, saya bisa menghentikan loop dan bahkan mengembalikan nilai, seperti yang terjadi pada loop berikutnya, di mana variabel counter dikalikan 4 hingga lebih dari 100, dan kemudian keluar dari loop sambil mengembalikan nilai counter.

  While loop digunakan untuk mengulang blok kode selama kondisi tertentu terpenuhi. Dalam contoh, saya menggunakan while num < 10 untuk mencetak "Hello there!" sampai nilai num mencapai 10. Setelah setiap iterasi, nilai num ditambah 1, sehingga loop akhirnya berhenti.

  For loop adalah cara yang lebih sering digunakan di Rust untuk iterasi melalui koleksi. Saya menggunakan for item in shopping_list.iter() untuk mengiterasi elemen-elemen dalam array shopping_list dan mencetak tiap item. Selain itu, saya juga menggunakan range notation (0..10) untuk mengiterasi angka dari 0 hingga 9. Range ini secara otomatis menghasilkan iterator yang mencetak angka satu per satu dari 0 sampai 9.

Secara keseluruhan, Rust menyediakan fleksibilitas dalam penggunaan loop untuk mengulang tindakan berdasarkan kondisi atau koleksi data, dengan opsi untuk mengendalikan kapan dan bagaimana loop tersebut berhenti.

- HashMap
  Dalam contoh program ini, saya mempelajari penggunaan HashMap di Rust, yang memungkinkan kita untuk menyimpan data dalam bentuk pasangan kunci-nilai (key-value). Program dimulai dengan membawa definisi HashMap dari pustaka standar Rust menggunakan perintah use std::collections::HashMap.

  Saya kemudian membuat hash map kosong menggunakan HashMap::new(). Setelah itu, saya menambahkan elemen-elemen ke dalam hash map menggunakan metode insert(<key>, <value>), di mana saya memasukkan beberapa pasangan kunci dan nilai berupa string. Contohnya, kunci "One" memiliki nilai "Book", kunci "Two" memiliki nilai "Keyboard", dan seterusnya.

  Untuk mengambil nilai dari hash map berdasarkan kunci, saya menggunakan metode get(<key>), yang mengembalikan nilai yang terkait dengan kunci tersebut jika ada. Jika kunci ditemukan, nilai yang terkait akan dicetak; jika tidak, maka hasilnya adalah None.

  Selain itu, saya juga menggunakan metode remove(<key>) untuk menghapus entri berdasarkan kunci tertentu dari hash map. Setelah menghapus entri dengan kunci "Three", saya mencoba mengaksesnya lagi, dan hasilnya adalah None, karena entri tersebut telah dihapus.

  Secara keseluruhan, HashMap di Rust adalah struktur data yang sangat berguna untuk menyimpan dan mengelola pasangan kunci-nilai secara efisien, dengan kemampuan untuk menambah, mengakses, dan menghapus elemen dengan cepat.

## Analisis Kode Robotika

- **Robotik dengan sistem Event-Driven**
  Dalam program ini, saya mempelajari cara menggunakan multi-threading dan channel communication di Rust untuk menangani berbagai jenis event pada sebuah robot. Program dimulai dengan mendefinisikan enum Event yang mencakup beberapa jenis event, seperti deteksi rintangan, perubahan tujuan, dan keadaan idle. Kemudian, dibuat struktur Robot yang memiliki nama dan fungsi untuk menangani event dengan metode handle_event. Tergantung pada jenis event yang diterima, robot akan menghindari rintangan, menuju tujuan baru, atau berada dalam keadaan idle.

  Simulasi lingkungan dilakukan dengan membuat fungsi environment_simulation yang mengirimkan event secara acak ke saluran komunikasi menggunakan Sender. Program utama menjalankan thread terpisah untuk mensimulasikan lingkungan, sementara thread utama menerima event dan memprosesnya dalam loop. Fungsi mpsc::channel() digunakan untuk komunikasi antar thread, di mana satu thread mengirimkan event dan thread lainnya menerima serta memprosesnya.

Program ini menunjukkan bagaimana cara menggunakan komunikasi antar thread dan menangani event secara real-time di Rust, yang dapat diterapkan dalam pengembangan sistem robotik atau aplikasi berbasis event lainnya.

- **Perencanaan Jalur Sederhana**
  Program ini mengimplementasikan algoritma A* untuk mencari jalur optimal pada grid dua dimensi, dengan mempertimbangkan biaya perjalanan (g(n)) dan perkiraan jarak ke tujuan (h(n)) menggunakan heuristik Manhattan. Algoritma ini menggunakan BinaryHeap untuk mengelola node yang diprioritaskan berdasarkan nilai f(n), yang merupakan jumlah dari biaya dan heuristik. Grid 5x5 digunakan untuk menguji jalur dari titik (0, 0) ke (4, 4), dengan nilai 0 sebagai ruang yang bisa dilalui dan 1 sebagai rintangan. Algoritma A* memastikan pencarian jalur yang lebih efisien dan optimal.

- **Gerakan robot dengan input pengguna**
  Program ini mengimplementasikan kontrol gerakan robot berdasarkan input pengguna melalui terminal. Robot mulai pada posisi (0, 0) dan dapat digerakkan ke atas, bawah, kiri, atau kanan menggunakan tombol w, s, a, dan d secara berulang dalam loop. Program juga memastikan bahwa robot tidak bergerak keluar dari batas (misalnya, tidak bisa bergerak lebih jauh ke bawah dari posisi 0). Input q digunakan untuk keluar dari program, dan posisi akhir robot ditampilkan sebelum program berakhir. Program ini memberikan pengalaman interaktif untuk mengontrol posisi robot secara langsung.

- **Simulasi Robot Menghindari Rintangan**
  Program ini menerapkan algoritma A* (A-star) untuk simulasi robot yang menghindari rintangan dalam grid 2D. Robot dimulai pada posisi (0, 0) dan bertujuan mencapai posisi (4, 4). Grid terdiri dari nilai 0 yang menunjukkan area bebas rintangan dan 1 yang menunjukkan rintangan. Algoritma A* digunakan untuk mencari jalur terpendek dari posisi start ke tujuan dengan mempertimbangkan biaya perjalanan (g-score) dan estimasi jarak ke tujuan (heuristic).

  Program ini menggunakan BinaryHeap untuk mengelola open set yang berisi node yang akan diproses, dan HashMap untuk melacak g-score serta jalur yang telah dilalui. Fungsi heuristic menghitung estimasi jarak antara dua titik menggunakan rumus jarak Manhattan. Program menghindari area yang terblokir dan terus mencari jalur alternatif sampai tujuan tercapai.

  Jika jalur ditemukan, program mencetak langkah-langkah yang diambil robot untuk mencapai tujuan. Jika tidak ada jalur yang ditemukan (misalnya, jika rintangan terlalu padat), program akan menginformasikan pengguna bahwa tidak ada jalur yang tersedia. Algoritma ini efektif untuk simulasi navigasi robot yang memerlukan penghindaran rintangan dalam lingkungan yang dinamis.

- **Penjadwalan Robot dengan Prioritas**
  Program ini mengimplementasikan penjadwalan tugas robot menggunakan sistem prioritas. Robot yang diberi nama "Atlas" memiliki antrean tugas yang diatur menggunakan BinaryHeap, di mana setiap tugas memiliki prioritas yang lebih tinggi diproses terlebih dahulu. Struktur Task menyimpan informasi tentang tugas, seperti

nama, prioritas, dan deskripsi tugas. Implementasi Ord dan PartialOrd memastikan bahwa tugas dengan prioritas lebih tinggi diproses terlebih dahulu oleh heap.

Robot memiliki dua status: Idle dan Busy. Ketika tugas baru ditambahkan, tugas tersebut akan disimpan dalam antrean dengan prioritas yang sesuai. Fungsi process_tasks akan memproses tugas berdasarkan urutan prioritas dan mengubah status robot menjadi Busy selama pemrosesan. Setelah semua tugas selesai, robot kembali ke status Idle.

Pada fungsi main, robot "Atlas" diberi tiga tugas dengan prioritas yang berbeda, dan tugas-tugas tersebut diproses sesuai urutan prioritas. Program mensimulasikan pemrosesan tugas dengan tidur selama satu detik untuk setiap tugas. Sistem ini berguna untuk penjadwalan robot dalam lingkungan yang membutuhkan pengelolaan prioritas tugas secara efisien.

- **Robot dengan Model Probabilistik**
  Program ini mengimplementasikan pencarian jalur probabilistik untuk robot menggunakan graf yang berisi node dan edge dengan jarak antar node. Setiap node diwakili oleh titik dalam ruang 2D (struktur Point), dan graf (Graph) terdiri dari beberapa node yang terhubung dengan edge yang memiliki jarak tertentu.

  Dalam algoritma pencarian jalur, probabilitas ketidakpastian pada setiap langkah dipertimbangkan untuk mensimulasikan kondisi dunia nyata di mana jarak yang ditempuh bisa bervariasi. Setiap kali robot bergerak ke node tetangga, sebuah faktor ketidakpastian (uncertainty) antara 0.9 dan 1.1 diterapkan pada jarak antar node, menghasilkan jarak yang sedikit berbeda setiap kali dijalankan. Hal ini mencerminkan ketidakpastian dalam pengukuran atau kondisi jalan yang berubah-ubah.

  Algoritma yang digunakan adalah varian dari algoritma pencarian jalur dengan menggunakan BinaryHeap (sebagai priority queue) untuk memilih jalur dengan biaya terendah (cost) ke node tujuan. Biaya untuk mencapai node tetangga dihitung berdasarkan jarak yang ditambah dengan faktor ketidakpastian. Jika biaya baru lebih rendah dari biaya yang sudah dihitung sebelumnya, maka jalur tersebut diprioritaskan.

  Pada fungsi main, graf dibuat dengan empat node dan beberapa edge, kemudian dijalankan pencarian jalur probabilistik dari node 1 ke node 4. Hasil pencarian berupa urutan node yang membentuk jalur optimal yang terhitung dengan ketidakpastian, atau pesan bahwa jalur tidak ditemukan jika tidak ada jalur yang memenuhi kriteria.

  Program ini mencerminkan pendekatan yang lebih realistis dalam perencanaan jalur robot, di mana kondisi dunia nyata sering kali mengandung ketidakpastian, seperti rintangan yang tidak terdeteksi atau perubahan di medan tempuh.