Analisis Week 14

Nama : Bara Khrisna Rakyan Nismara

NIM : 1103210151

Kelas : TK45G09

Arrays and Vectors

Kode ini mendemonstrasikan penggunaan array dan vektor dalam bahasa pemrograman Rust. Array, seperti yang ditunjukkan oleh working_days dan working_days_num, adalah koleksi nilai dengan tipe data yang sama yang disimpan secara berurutan di memori. Array memiliki ukuran tetap yang ditentukan saat kompilasi, dan elemen-elemennya dapat diakses menggunakan indeks berbasis nol. Kode ini menunjukkan inisialisasi array secara langsung dengan nilai dan inisialisasi dengan nilai yang sama berulang kali. Vektor, di sisi lain, seperti yang ditunjukkan oleh nephews_age, zeroes, names, dan fruit, adalah koleksi dinamis yang dapat bertambah atau berkurang ukurannya pada saat runtime. Vektor dapat diinisialisasi dengan nilai atau dibuat kosong lalu ditambahkan nilai menggunakan metode push(). Elemen vektor juga dapat diakses menggunakan indeks dan dimodifikasi. Selain itu, kode ini menunjukkan metode pop() untuk menghapus elemen terakhir dari vektor.

Secara keseluruhan, kode ini memberikan contoh dasar tentang cara membuat, menginisialisasi, mengakses, dan memanipulasi array dan vektor di Rust. Array berguna ketika kita tahu ukuran koleksi pada waktu kompilasi dan ukuran tersebut tidak akan berubah. Sedangkan vektor lebih fleksibel karena dapat mengubah ukuran dan merupakan pilihan yang lebih tepat ketika kita tidak mengetahui jumlah item di dalam koleksi pada saat kompilasi. Kode juga menyoroti perbedaan mendasar antara kedua tipe data ini, termasuk sintaks deklarasi dan opsi manipulasi yang tersedia untuk setiap jenis. Dengan berbagai macam contoh, kode ini secara efektif mengilustrasikan konsep-konsep kunci ini, menyediakan gambaran praktis untuk pemahaman penggunaan array dan vektor di Rust.

Data Types

Kode ini mendemonstrasikan berbagai konsep dasar dalam bahasa pemrograman Rust, termasuk definisi struct, tipe data primitif, variabel, dan output ke konsol. Kode ini memperkenalkan dua jenis struct: struct klasik (Student) yang memiliki nama field dan tipe datanya, dan tuple struct (Grades) yang hanya memiliki tipe data. Fungsi main berfungsi sebagai titik masuk program dan berisi berbagai contoh penggunaan tipe data seperti integer (u32), float (f32), boolean (bool), karakter (char), dan string (&str dan String). Kode tersebut menyoroti perbedaan antara variabel immutable dan mutable, variable shadowing, dan inferensi tipe data oleh compiler Rust.

Selain itu, kode ini menunjukkan cara membuat tuple dan mengakses elemen-elemennya berdasarkan indeks, serta cara membuat instance struct dan mengakses field struct klasik dan tuple struct.
Lebih lanjut, kode ini mengilustrasikan penggunaan makro println! untuk mencetak output ke konsol, termasuk pemformatan string dengan substitusi nilai. Ini juga menjelaskan fitur-fitur Rust seperti variable binding, variabel yang bisa diubah (mut), dan variable shadowing yang memungkinkan variable dengan nama yang sama di dalam scope yang berbeda. Dengan berbagai contoh, kode ini memberikan panduan praktis untuk memahami struktur dan tipe data dasar di Rust dan merupakan fondasi yang baik untuk mempelajari konsep-konsep yang lebih kompleks dalam bahasa tersebut. Kode ini juga mencakup bagaimana membuat variable dan menginisialisasinya, serta bagaimana cara mengakses nilai dari variable-variable tersebut.

Hash Map

Analisis Paragraf 1: Inisialisasi dan Pengisian HashMap

Paragraf kode ini menunjukkan proses inisialisasi dan pengisian sebuah hash map di Rust. Pertama, kode ini mengimpor tipe data HashMap dari pustaka standar Rust menggunakan use std::collections::HashMap;. Kemudian, sebuah hash map kosong bernama items dibuat, di mana key dan value keduanya bertipe String. Kode ini lalu menambahkan tiga pasangan key-value ke dalam hash map menggunakan metode insert(). Dalam contoh ini, key merepresentasikan angka ("One", "Two", "Three") yang berupa string dan value merepresentasikan item ("Book", "Keyboard", "Sunglasses") yang juga berupa string. Paragraf ini berfokus pada cara membuat dan mengisi data ke dalam hash map, menunjukkan bagaimana key dan value disimpan dan bagaimana data tersebut diasosiasikan satu sama lain.

Analisis Paragraf 2: Akses dan Penghapusan Data dari HashMap

Paragraf kode ini menunjukkan bagaimana cara mengakses dan menghapus data dari hash map. Kode ini pertama-tama mengambil value yang terkait dengan key "Two" menggunakan metode get(), dan kemudian mencetak hasil yang berupa Some("Keyboard") menggunakan format debug {:?}. Ini menunjukkan bagaimana kita dapat mengambil data tertentu dari hash map berdasarkan key yang kita ketahui. Selanjutnya, kode ini menghapus entri yang terkait dengan key "Three" menggunakan metode remove(). Setelah penghapusan, kode mencoba untuk mengambil value yang terkait dengan key "Three" menggunakan metode get(). Karena entri tersebut telah dihapus, kode akan mencetak None, yang menunjukkan bahwa entri tersebut tidak lagi ada dalam hash map. Paragraf ini berfokus pada operasi yang berbeda setelah data dimasukkan, yaitu mengambil dan menghapus data dari hash map, dan juga menunjukan bagaimana hash map mengelola data yang tidak ada.

Hello World

Kode ini adalah program Rust yang sangat sederhana. Fungsinya adalah mencetak teks "Hello, world!" ke konsol.
Analisis Singkat:

fn main() { ... }: Ini adalah fungsi utama (main function), titik awal eksekusi program Rust. Setiap program Rust harus memiliki fungsi main.

println!("Hello, world!");: Ini adalah baris kode yang melakukan pencetakan ke konsol. println! adalah makro (bukan fungsi) yang digunakan untuk mencetak teks dengan baris baru di akhir. Teks yang akan dicetak adalah "Hello, world!".

Kesimpulan:
Program ini adalah contoh klasik "Hello, world!" yang biasa digunakan untuk memperkenalkan sebuah bahasa pemrograman. Ini menunjukkan struktur dasar program Rust, dan bagaimana cara mencetak output ke konsol.

If Else

Paragraf 1: Penggunaan Dasar dan Ekspresi if/else
Kode ini dimulai dengan demonstrasi dasar dari pernyataan if/else dalam Rust. Pernyataan if 1 == 2 akan selalu bernilai false karena 1 tidak sama dengan 2, sehingga kode di dalam blok else, yaitu println!("The numbers are not equal"), akan dieksekusi. Hal ini menunjukkan bagaimana alur kontrol program dapat bercabang berdasarkan kondisi boolean. Selanjutnya, kode ini mendemonstrasikan bahwa if/else dapat digunakan sebagai ekspresi untuk mengikat nilai ke variabel. Variabel sunny_day diinisialisasi sebagai true. Kemudian, nilai variabel take_jacket ditentukan berdasarkan nilai sunny_day. Jika sunny_day bernilai true, take_jacket akan bernilai "Don't take a jacket", dan sebaliknya, jika false maka akan bernilai "Take a jacket". Hal ini menggambarkan penggunaan if/else sebagai ekspresi yang menghasilkan nilai dan bukan sekadar pernyataan alur kontrol. Output dari ekspresi ini, yaitu nilai take_jacket, kemudian dicetak ke konsol.
Paragraf 2: Rantai Kondisional if/else if/else
Paragraf kedua dari kode ini menunjukkan penggunaan rantai kondisional if/else if/else untuk mengevaluasi beberapa kondisi. Variabel num diinisialisasi dengan nilai 100 dan out_of_range sebagai boolean yang tidak diinisialisasi. Serangkaian kondisi dievaluasi secara berurutan. Jika num kurang dari 0, maka out_of_range akan bernilai true. Jika tidak, kode memeriksa apakah num sama dengan 0. Jika kedua kondisi ini salah, akan dilakukan pengecekan berikutnya apakah num lebih dari 101. Jika salah satu dari ketiga kondisi ini benar, maka out_of_range diatur menjadi true. Jika tidak ada kondisi yang benar (dalam kasus ini, karena 100 tidak kurang dari 0, sama dengan 0, dan tidak lebih besar dari 101), maka out_of_range diatur menjadi false. Pada

akhirnya, nilai dari variabel out_of_range dicetak ke konsol, yang akan menghasilkan false. Ini menunjukkan bagaimana if/else if/else digunakan untuk menangani beberapa kondisi yang saling eksklusif dan bagaimana memberikan nilai berdasarkan kondisi yang terpenuhi.

Loops

Paragraf 1: Penggunaan loop dan while dalam Rust

Kode ini memperkenalkan dua jenis loop dasar dalam Rust, yaitu loop dan while. Awalnya, terdapat contoh loop yang dikomentari, yang jika tidak dikomentari akan menghasilkan infinite loop, terus menerus mencetak "Hello!". Ini menekankan bahwa loop berjalan tanpa henti kecuali ada pernyataan break. Kemudian, kode memberikan contoh penggunaan loop dengan break, di mana variabel counter dikalikan dengan 4 dalam setiap iterasi. Loop akan berhenti ketika counter lebih besar dari 100, dan nilai counter saat itu akan di-break dan diikat ke variabel loop_stop. Hal ini menunjukkan bahwa loop dapat menghasilkan nilai saat loop dihentikan. Selanjutnya, kode mendemonstrasikan while loop yang berjalan selama kondisi num < 10 terpenuhi. Dalam setiap iterasi, ia mencetak "Hello there!" dan meningkatkan nilai num hingga akhirnya loop berhenti. Bagian ini menjelaskan bagaimana loop dan while bekerja, termasuk penggunaan break untuk menghentikan loop dan cara while bekerja berdasarkan kondisi.

Paragraf 2: Penggunaan for loop dengan iterator

Paragraf kedua dari kode berfokus pada penggunaan for loop dengan iterator. Kode ini pertama kali mendefinisikan sebuah array string bernama shopping_list. Kemudian, ia menggunakan for item in shopping_list.iter() untuk mengiterasi melalui setiap elemen dalam array tersebut, mencetak setiap item belanja. Metode iter() digunakan untuk membuat iterator dari array tersebut, dan for loop mengikat setiap nilai dari iterator ke variabel item. Selanjutnya, kode ini mendemonstrasikan cara membuat iterator menggunakan rentang angka, yaitu 0..10. for number in 0..10 akan melakukan iterasi dari 0 hingga 9 (nilai 10 tidak termasuk), mencetak setiap angka yang dihasilkan. Bagian ini menunjukkan cara yang efektif untuk melakukan iterasi melalui koleksi data dan rentang angka dengan for loop, menekankan penggunaan iterator yang disediakan oleh Rust.

Perencanaan jalur

Paragraf 1: Penggunaan loop dan while dalam Rust

Kode ini memperkenalkan dua jenis loop dasar dalam Rust, yaitu loop dan while. Awalnya, terdapat contoh loop yang dikomentari, yang jika tidak dikomentari akan menghasilkan infinite loop, terus menerus mencetak "Hello!". Ini menekankan bahwa loop berjalan tanpa henti kecuali ada pernyataan break. Kemudian, kode memberikan contoh penggunaan loop dengan break, di mana variabel counter dikalikan dengan 4 dalam setiap iterasi. Loop akan berhenti ketika counter lebih besar dari 100, dan nilai counter saat itu akan di-break dan diikat ke variabel loop_stop. Hal ini menunjukkan bahwa loop dapat menghasilkan nilai saat loop dihentikan. Selanjutnya,

kode mendemonstrasikan while loop yang berjalan selama kondisi num < 10 terpenuhi. Dalam setiap iterasi, ia mencetak "Hello there!" dan meningkatkan nilai num hingga akhirnya loop berhenti. Bagian ini menjelaskan bagaimana loop dan while bekerja, termasuk penggunaan break untuk menghentikan loop dan cara while bekerja berdasarkan kondisi.

Paragraf 2: Penggunaan for loop dengan iterator

Paragraf kedua dari kode berfokus pada penggunaan for loop dengan iterator. Kode ini pertama kali mendefinisikan sebuah array string bernama shopping_list. Kemudian, ia menggunakan for item in shopping_list.iter() untuk mengiterasi melalui setiap elemen dalam array tersebut, mencetak setiap item belanja. Metode iter() digunakan untuk membuat iterator dari array tersebut, dan for loop mengikat setiap nilai dari iterator ke variabel item. Selanjutnya, kode ini mendemonstrasikan cara membuat iterator menggunakan rentang angka, yaitu 0..10. for number in 0..10 akan melakukan iterasi dari 0 hingga 9 (nilai 10 tidak termasuk), mencetak setiap angka yang dihasilkan. Bagian ini menunjukkan cara yang efektif untuk melakukan iterasi melalui koleksi data dan rentang angka dengan for loop, menekankan penggunaan iterator yang disediakan oleh Rust.

Gerakan Robot

Kode ini mendefinisikan sebuah program simulasi pergerakan robot sederhana di mana robot dapat bergerak ke atas, bawah, kiri, atau kanan berdasarkan perintah pengguna. Program dimulai dengan mendefinisikan struktur data Robot yang memiliki koordinat x dan y sebagai i32. Implementasi dari Robot termasuk fungsi new untuk membuat robot di posisi awal (0,0), move_robot untuk mengubah posisi robot berdasarkan input perintah, dan get_position untuk mengambil posisi robot saat ini. Dalam fungsi main, program menginisialisasi objek Robot, mencetak posisi awal, dan kemudian masuk ke dalam loop utama. Dalam loop, program meminta input perintah dari pengguna, membaca perintah tersebut, dan kemudian memeriksa apakah perintah tersebut adalah "exit". Jika iya, program akan mencetak posisi akhir robot dan keluar dari loop. Jika tidak, program akan memanggil fungsi move_robot untuk mengubah posisi robot dan kemudian mencetak posisi robot saat ini. Program ini terus berulang sampai perintah "exit" dimasukkan, memungkinkan pengguna untuk mengontrol pergerakan robot dan melihat posisinya di setiap langkah.

Simulasi Robot

Kode ini mengimplementasikan algoritma Breadth-First Search (BFS) untuk mencari jalur terpendek pada grid dua dimensi. Kode dimulai dengan mendefinisikan struct Point untuk merepresentasikan koordinat, termasuk fungsi new untuk memudahkan pembuatan instance Point. Fungsi is_valid digunakan untuk mengecek apakah suatu koordinat valid untuk dijelajahi, berdasarkan batasan grid, nilai grid (0 menandakan jalur terbuka), dan status kunjungan. Fungsi find_path adalah inti algoritma BFS, menggunakan VecDeque sebagai queue untuk menyimpan titik-titik yang akan dieksplorasi, dan vector parent untuk merekonstruksi jalur setelah menemukan

tujuan. Pencarian dimulai dari titik awal, dan secara iteratif mengeksplorasi tetangga titik saat ini yang memenuhi syarat. Jika titik tujuan ditemukan, jalur akan dikembalikan melalui rekursi parent, jika tidak maka fungsi akan mengembalikan None. Fungsi main mendefinisikan sebuah grid sampel, titik awal, dan titik tujuan, kemudian memanggil fungsi find_path. Hasil pencarian kemudian di-match, dan jika ada jalur ditemukan, maka jalur tersebut akan dicetak dengan langkah-langkahnya. Jika tidak ada jalur, pesan tidak ada jalur akan dicetak.

Penjadwalan Robot

Kode ini mengimplementasikan antrean prioritas menggunakan BinaryHeap dari pustaka standar Rust, di mana tugas dengan prioritas tertinggi akan diproses terlebih dahulu. Pertama, struct Task didefinisikan untuk merepresentasikan tugas, yang terdiri dari prioritas (i32) dan deskripsi (String). Implementasi dari trait Ord, PartialOrd, dan PartialEq memungkinkan Task untuk digunakan dalam BinaryHeap. Khususnya, implementasi Ord membandingkan tugas berdasarkan prioritas secara terbalik, sehingga BinaryHeap berperilaku sebagai max-heap (tugas dengan prioritas tertinggi di bagian atas). Dalam fungsi main, sebuah BinaryHeap dibuat untuk menyimpan tugas, kemudian beberapa tugas ditambahkan ke dalam queue dengan prioritas berbeda. Loop while let kemudian digunakan untuk mengambil tugas dari queue satu per satu, selalu mengambil tugas dengan prioritas tertinggi, dan mencetak deskripsi tugas yang diselesaikan. Proses ini menunjukkan bagaimana BinaryHeap dapat digunakan untuk membuat antrean prioritas di mana tugas diselesaikan berdasarkan prioritas yang telah ditentukan.

Sistem Event

Kode ini mensimulasikan perilaku robot yang bereaksi terhadap berbagai event menggunakan struktur data enum untuk merepresentasikan tipe event yang mungkin terjadi dan struct untuk merepresentasikan robot dengan posisi dan tujuan. Struct Point merepresentasikan koordinat dengan metode new untuk membuat instance dengan mudah. Enum Event mendefinisikan tiga tipe event yaitu ObstacleDetected, GoalChanged, dan Idle, yang memungkinkan robot untuk merespon kondisi yang berbeda. Struct Robot menyimpan posisi dan tujuan robot, dan memiliki metode new untuk membuat robot dengan posisi dan tujuan awal, handle_event untuk menanggapi tipe event yang berbeda, avoid_obstacle untuk memodifikasi posisi robot saat rintangan terdeteksi, dan move_to_goal untuk menggerakkan robot ke tujuan. Dalam fungsi main, objek Robot dibuat, queue VecDeque diinisialisasi dengan beberapa event, kemudian program melakukan iterasi melalui event yang ada. Setiap event akan diproses oleh handle_event untuk mengubah perilaku robot. Simulasi ini menunjukkan bagaimana robot dapat merespon event yang berbeda dan memodifikasi perilakunya sesuai dengan event tersebut.

Robot Probabilitas

Kode ini mensimulasikan perilaku robot yang bereaksi terhadap berbagai event menggunakan struktur data enum untuk merepresentasikan tipe event yang mungkin terjadi dan struct untuk merepresentasikan robot

dengan posisi dan tujuan. Struct Point merepresentasikan koordinat dengan metode new untuk membuat instance dengan mudah. Enum Event mendefinisikan tiga tipe event yaitu ObstacleDetected, GoalChanged, dan Idle, yang memungkinkan robot untuk merespon kondisi yang berbeda. Struct Robot menyimpan posisi dan tujuan robot, dan memiliki metode new untuk membuat robot dengan posisi dan tujuan awal, handle_event untuk menanggapi tipe event yang berbeda, avoid_obstacle untuk memodifikasi posisi robot saat rintangan terdeteksi, dan move_to_goal untuk menggerakkan robot ke tujuan. Dalam fungsi main, objek Robot dibuat, queue VecDeque diinisialisasi dengan beberapa event, kemudian program melakukan iterasi melalui event yang ada. Setiap event akan diproses oleh handle_event untuk mengubah perilaku robot. Simulasi ini menunjukkan bagaimana robot dapat merespon event yang berbeda dan memodifikasi perilakunya sesuai dengan event tersebut.