Title:	RUST
Student Name:	FATIQ HUSSNAIN
Registration No:	2021-CS-140

How to do this task?

Goal of this task is to get all alliable interview questions related to your topic.

Save file name with your topic name.

Search different keywords related to your topic on the google and copy questions available on your topic from at least 20 websites. Do not use the ChatGPT or any text generative tool for this task please. If website has pagination go throw the pagination and copy all contents.

For example, the Technology is Java so you can search like
Java Interview Questions, Java Advance Interview Questions,
Java Interview Guide, Java Interview materials, Java Interview preparation.

Sr.	Website Link from where you took questions.
1	https://www.rust-lang.org/
2	https://www.turing.com/interview-questions/rust
3	https://zerotomastery.io/blog/rust-interview-questions-and-answers/
4	https://coderpad.io/interview-questions/rust-interview-questions/
5	https://101blockchains.com/top-rust-interview-questions/
6	https://codesubmit.io/interview/rust-interview-questions
7	https://diptendud.medium.com/rust-interview-questions-answers-2023-part-1-d67d8fd5da00
8	https://www.usebraintrust.com/hire/interview-questions/rust-developers
9	https://simpleprogrammer.com/rust-interview-questions-answers/
10	https://www.fullstack.cafe/blog/rust-interview-questions
11	https://javarevisited.blogspot.com/2022/07/rust-interview-questions-with-answers.html
12	https://github.com/imhq/rust-interview-handbook
13	https://optymize.io/rust-interview-questions/
14	https://mentorcruise.com/questions/rust/
15	https://mindmajix.com/rust-interview-questions
16	https://www.devopsschool.com/blog/top-50-rust-programming-interview-questions-and-answers/
17	https://users.rust-lang.org/t/rust-interview-questions/12670
18	https://interviewing.io/rust-interview-questions

19	https://www.remoterocketship.com/advice/10-rust-interview-questions-and-answers-in-2023
20	https://www.firehire.ai/interview-questions/rust

Template

Website Link1

All contents taken from link1

Question 1: What is Rust Programming Language?

Answer:

Rust is a general-purpose programming language emphasizing performance, type safety, and concurrency. It enforces memory safety, meaning that all references point to valid memory.

Question 2: Why Rust ?

Answer:

- **Performance**
- **Reliability**
- **Productivity**

Question 3: What things can we build in Rust?

Answer:

- Command Line
- Web Assembly
- Networking
- Embedded

Question 4: What things can we build in Rust?

Answer:

- Command Line
- Web Assembly
- Networking
- Embedded

Question 5: Describe ownership in rust?

Answer: In Rust, ownership is a fundamental concept that defines and governs how the memory is managed in a Rust program. It is a mechanism that allows Rust to implement memory safety without needing a garbage collector.

Every value, in Rust, has an owner, the variable that holds the value. When the owner goes out of scope, the value is dropped, which frees the associated memory.

Question 6: How to declare global variables in rust?

Answer: In Rust, you can declare a global variable using the static keyword. The static keyword declares a global variable with a static lifetime, which means that it exists for the entire duration of the program's execution.

To declare a global variable, you need to specify the type, and it must have a constant expression for initialization. Additionally, since global variables can be accessed from multiple threads, one must ensure to handle synchronization when using mutable global variables.

Question 7: What are the limitations of rust?

Answer: Here are some major limitations associated with the Rust programming language:

- **Learning Curve**: Rust can be difficult, especially for those new to programming or unfamiliar with systems programming languages. It has a steep learning curve, and its syntax can be complex and unfamiliar to many developers.
- **Memory Management**: While Rust's ownership and borrowing system is designed to prevent memory-related bugs, it can also be quite restrictive,

requiring developers to manage memory usage and ownership of variables carefully.

- **Slow Compilation**: Rust is known for having slow compilation times, especially compared to other modern programming languages. This can frustrate developers who need to iterate quickly during the development process.
- **Limited Libraries**: Rust is still a relatively new language, and as a result, its library ecosystem is not as mature as that of other languages like Python or JavaScript. This can make it difficult to find and use third-party libraries and frameworks.

Q8. Why does Rust have high performance ?

Answer: Rust has high performance because it compiles directly to native machine code. This enables the program to run at full speed since there isn't an interpreter needed to translate the program to machine instructions.

Q9: Explain the concept of ownership and borrowing in the context of managing mutable data in Rust.

Answer: In Rust, ownership and borrowing play a significant role in managing mutable data efficiently and safely.

Ownership ensures that a piece of data has a single owner at any given time. When a value is owned by a variable, that variable has exclusive control over the data. When the owner goes out of scope, Rust automatically deallocates the data, preventing memory leaks. To mutate data, you must have ownership of the data.

Borrowing, on the other hand, allows temporary and read-only access to data without transferring ownership. Borrowing allows multiple references to access the data simultaneously, but only one mutable reference can be active at a time. Borrowing is enforced through strict rules at compile-time, which prevent data races and other memory safety issues.

By combining ownership and borrowing, Rust enables safe and efficient mutation of data without the need for a garbage collector.

Q10: What are Cargo and Cargo.lock in Rust?

Answer: Cargo is a development system and package manager developed for Rust users that enables direct management of projects from the language. Cargo helps in creating your code in Rust alongside downloading the libraries required for your code and creating the libraries according to your use case. Cargo.lock is the file created when a user implements the cargo build command. The command automatically generates the cargo.lock file for maintaining a tab on all dependencies associated with the user application.

Q11. How safe is Rust compared to C and C++?

Answer: One of the main benefits of Rust in comparison to C is the ability to write safe code. Rust doesn't, like C does, require memory management or performing pointer arithmetic. In comparison to C++, Rust offers safety by not showing arbitrary behavior if a mistake is made.

Q12: What are the key features of Rust?

Answer: Rust's key features include:

- Memory safety without a garbage collector.
- Zero-cost abstractions for high-level programming.
- Concurrency support with ownership, borrowing, and lifetimes.
- A strong, static type system.
- Minimal runtime.

Q13: What is "lifetimes" in Rust?

Answer: Lifetimes specify how long a reference to data should remain valid, preventing "dangling references" or accessing data that might be cleaned up.

Q14: What is the difference between String and &str in Rust?

Answer: String is a growable, heap-allocated string type, while &str is a string slice pointing to a string in memory.

Q15: What will be the output of the code below?

```
fn main() {
    let x = 5;
    let y = x;
    println!("x: {}, y: {}", x, y);
}
```

Answer:

```
x: 5, y: 5
```

Q16: What is a trait in Rust?

Answer: A trait defines shared behavior that types can implement. It is similar to interfaces in other languages. For example:

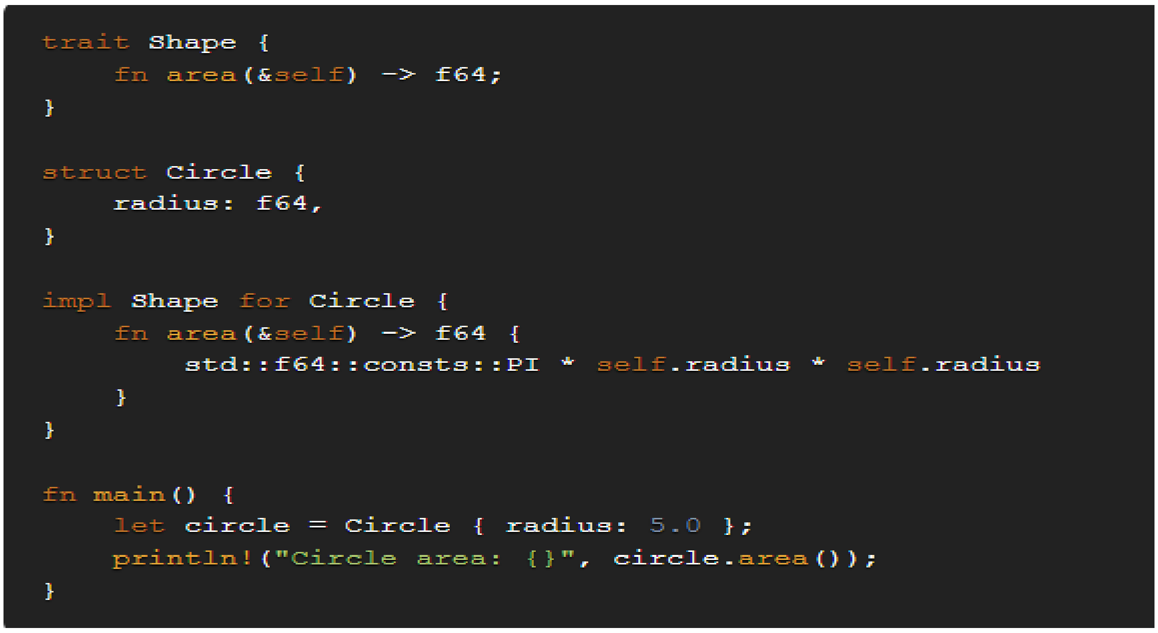
```
trait Speak {
    fn speak(&self);
}

struct Dog;

     Speak for Dog {
    fn speak(&self) {
        println!("Woof!");
    }
}
```

Q17: Does Rust support OOP ?

Answer: Yes why not.



```rust
trait Shape {
    fn area(&self) -> f64;
}

struct Circle {
    radius: f64,
}

impl Shape for Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * self.radius * self.radius
    }
}

fn main() {
    let circle = Circle { radius: 5.0 };
    println!("Circle area: {}", circle.area());
}
```

Q18: What are the programming paradigms supported by Rust?

Answer: The programming paradigms supported by Rust are

- Concurrent computing
- Functional programming
- Generic programming
- Structured programming

Q19: Copy vs Clone in Rust?

Answer: The Clone trait defines the ability to explicitly create a deep copy of an object T. When we call Clone for type T, it does all the arbitrarily complicated operations required to create a new T. The Copy trait in Rust defines the ability to implicitly copy an object. The behavior Copy is not overloadable. It is always a simple bitwise copy

Q20: What are the Error Handling procedures in Rust?

Answer: Rust Error Handling is categorized into three parts:

- **Recoverable Error with Results:** If an error occurs, the program doesn't stop completely. Instead, it can easily be interpreted or responded.
- **Unrecoverable Errors with Panic**: If something wrong goes with the code, Rust's panic macro comes into action, shows the error message, clean the error and then quit.

- **Panic or Not to Panic**: When you are dicey about calling panic or not, write the code that panics, and the process will continue as 2nd.