**Async Rust And Stabilization Of async/.await feature**

* **Summary:**

Async rust is a feature of Rust programming language in which task is done concurrently on a single thread which is called as Multitasking, where as multithreading is a mutual alike concept which is based on CPU intensive bounds, while async rust is based on Input/Output intensive bounds.In Async rust we let the I/O bounds computations to run at the same time because in that case when other task is awaiting , async rust let the other task to run that isn’t waiting.

In case when we need a result of asynchronous computations, we use async/.await feature.Async function doesn’t start running immediately you must have to use an executor or .await. In Rust, values that are awaitable are called ‘futures’. In async Rust we must need an external library as standard library doesn’t come with executor(that takes care of executing the futures).

When We Are writing application on async rust , We need to remember some basic steps,first we create new application using cargo new xyz(the name of application), next we add dependencies of our application,then we write our application code in main.rs file.

In Async Rust, Unlike a regular function, calling an async fn doesn't have any immediate effect. Instead, it returns a Future. This is a suspended computation that is waiting to be executed. To actually *execute* the future, we use the .await operator:

async fn another\_function() {

// Create the future:

let future = first\_function();

// Await the future, which will execute it (and suspend

// this function if we encounter a need to wait for I/O):

let result: u32 = future.await;

...

}

The above example shows the first difference between Rust and other languages: we write future.await instead of await future. This syntax integrates better with Rust's? operator for propagating errors (which, after all, are very common in I/O). You can simply write future.await ? to await the result of a future and propagate errors. It also has the advantage of making method chaining painless.