RUST PROGRAMMING LANGUAGE

I. HISTORICAL PERSPECTIVE OF RUST

- 2006 = Creation of the Rust programming language by mozilla employee, **Graydon Hoare**, and named it after the rust family of fungi.
- 2009-2010 = sponsoring of the project by **Mozilla**. Compile rustc creation written in OCaml.
- 2015 = The first and only Rust stable release was on **May 15, 2015** and was called the Rust 1.0.

II. LANGUAGE DESIGN GOALS AND LANGUAGE PARADIGMS

A. Design Goals

Graydon Hoare's design goal for Rust was implemented and created to run a faster, safer and practical systems programming language. Rust is made to improve a system that would be more abstract and efficient due to its focus on speed, more secured memory safety, highly concurrent support and more control that maintains boundaries than other programming language.

B. Language Paradigms

Rust is a multi-paradigm programming language which shows the concurrent-action, imperative and functional paradigms. It supports several multiple programming styles like object-oriented, functional, procedural, generic programming, parallel, and data abstraction.

III. LANGUAGE APPLICATION AREAS

Rust Programming Language is used by web developers and programmers as of today. It is commonly used for web browsers or web browsers engine like the project of Mozilla, Servo and Quantum. Its own compiler is also written in their own language. Rust is used as a file system in Dropbox. Rust is also progressing as a programming language for game developers like Rustation, a Rust Playstation emulator and Piston, which is a Rust game engine.

IV. LANGUAGE ENVIRONMENT AND TOOLS, LIBRARIES, FRAMEWORKS, DOCUMENTATION RESOURCES

A. **Environment** - one IDE that is used by Rust is the **RustDT**, it is Eclipse based. It has a source code editor and it is using Cargo (a package manager).
   Tools - Rust includes the following tools: compiler, **cargo**, debugging support, profiling, formatting of codes, editor and IDE support, testing, code completion, documentation, and checks spelling.
B. **Libraries** - The **Rust standard library** has different kinds of modules. It also characterizes the Rust Prelude which is a list of things that Rust can automatically import to every project or program.
C. **Frameworks** - **Rocket** is a web framework for Rust (nightly) while **Iron** is a middleware oriented framework for Rust.
D. **Documentation Resources** - *The Rust Programming Language* is the documentation resource of Rust. This book contains important topics for learning Rust. It has two editions; the second edition is still under construction.

V. MAIN LANGUAGE FEATURES AND CONSTRUCTS

A. Features of Rust

- **Zero-cost abstraction**- this can compile low-level languages and pays only for features that you actually use
- **Move semantics**- not every variable owns resources.
- **Guaranteed memory safety**- this helps to prevent a dereference to a null pointer, using a dangling pointer, you cannot forget to free memory and cannot attempt to free already freed memory.
- **Threads without data races**- it mostly prevents rust ownership system.
- **Trait based generics** - traits are bounded to generic structs.

- **Pattern matching** - it is quite common in rust. We can use them in variable bindings, match statements and others.
- **Minimal runtime** - minimized the time using the compiler.
- **Efficient C Bindings** - It makes it simple by the means of communicating C APIs.

B. Language Constructs

- **Keywords** - 35 currently used and 17 total reserved keywords.
- **Variables and Mutability** - adding "**mut**" in front of the variable name will make it mutable.
- **Constants** - values that has a name are not allowed to change. "mut" are not allowed and constants are always immutable.
- **Shadowing** - The second variable will shadow the first or the one before, meaning the second variable value is what we will see when we use the variable.
- **Data types** - Scalar and Compound
    - **Scalar** - integer types, floating point types, numeric operations, boolean type and character type.
    - **Compound** - we have the tuples and arrays.
- **Functions**
    - **Function parameters** - special variables which are part of the functions signature.
    - **Function bodies** - it is a compromised progression of series of statements that alternatively finishing an expression.
    - **Statement and Expressions**
        - Statements this are guidelines that plays activity and do not return values.
        - Expressions it evaluates into a resulting value.
- **Comments**
    - This is use to explain a line of code or make a comment using a the two slashes or //
- **Control Flow**
    - **If expressions** - it allows to branch the codes depending on the conditions.
    - **Multiple conditions with else if**- it is the combination of if and else.
    - **Using if in a let statement** - it can be actually use at the right side of our let statement.
    - **Repetition with loops**- a loop inside the body of the loop.
    - **Repeating code with loop**- it tells us to execute a block of code repetitively or until we stop the loop.
    - **Conditional loops** - it is a combination of if, else and break.
    - **Looping with for**- using the while construct over an elements of a collection or an array.