# RUST

A PREPRINT

**Louis Galluzzi, Steven Phu, louisgalluzzi@email.arizona.edu, sphu@email.arizona.edu**

April 7, 2020

**ABSTRACT**

## 1 Introduction

## 2 History

Rust was designed in 2006 by Graydon Hoare, who was an employee of Mozilla. The project began as a hobby project, but changed when Graydon presented the prototype to his manager. In 2009, Mozilla began sponsoring the project in order to rebuild their browser stack. The main reason for the development of Rust was to make systems programming safer by solving two main issues: memory management and concurrency.

On January 2012, the pre-alpha version of Rust compiler was released. The first stable version did not come out until May 15, 2015. Rust shifted to a self-hosting compiler named rustc in 2010.

Rust began to gain traction in 2015 by winning the third most loved language in Stack Overflow. Since then it has won first place every single year. Rust is still alive with the most stable version being released March 12, 2020. The reddit subcommunity has 94,500 users in it.

## 3 Control Structures

### 3.1 If

The "if" expression is pretty standard with the other languages. It starts with the keyword "if", which is then followed by a condition. If condition evaluates to true, the next block of code that is wrapped in curly brackets will be executed. There is an optional else expression that can be placed after the if block of code. This expression gives the program an alternative block of code that will execute if the "if" expression evaluates to false. If no else expression is provided the if block of code will be skipped. In the example picture below. We assign variable number to 372. The if expression checks if number is less than 372. Since the if expression evaluates to false, it goes to the else expression and prints out "condition was false".

```rust
pub fn main() {
    let number = 372;

    if number < 372 {
        println!("condition was true");
    } else {
        println!("condition was false");
    }
}
```

```
C:\Users\louis\hello_cargo\project>cargo run
   Compiling project v0.1.0 (C:\Users\louis\hello_cargo\project)
    Finished dev [unoptimized + debuginfo] target(s) in 0.34s
     Running `target\debug\project.exe`
condition was false
```

### 3.2 Else If

The "else if" expression is used in combination with if expression to allow multiple conditions. It begins with a regular if expression then is followed by an "else if" expression. Each "else if" expression is followed by a condition and a block of code. If the condition evaluates true then the block of code is executed. Otherwise it skips the block of code. Just like the if expression, there is an optional "else" expression.

```rust
pub fn main() {
    let number = 372;

    if number == 372 {
        println!("Comparative programming!");
    } else if number  == 473 {
        println!("Automata");
    } else if number == 335 {
        println!("Web programming");
    } else {
        println!("Not a class");
    }
}
```

### 3.3 If Let

Rust has an "if let" expression that allows you to combine if and let to handle values that match a pattern while ignoring the rest. The benefit of this expression is less typing, less indentation, and less code. In our example, we set condition to be true. We set class to equal the first true condition in the if else expression. In this case, class would be assigned 372.

```rust
pub fn main() {
    let condition = true;
    let class = if condition {
        372
    } else {
        473
    };

    println!("The value of number is: {}", class);
}
```

### 3.4 loop

The loop expression allows Rust to execute a code infinitely until the user explicitly tells it to stop. The example below will print the statement "Comparative!" infinitely as there is no break or return.

```rust
pub fn main() {
    loop {
        println!("Comparative!");
    }
}
```

### 3.5 Return

Rust has the normal return just like any other languages where it'll return a value from a method with the return expression. Like python Rust will default return None if a return expression does not exist. Rust has another way to return a value from loops with the expression break. In order to achieve this we declare a variable and assign it to loop. Once a break condition has been met, we put the value we want returned after the break. In the example below, we have result hold the return value from the loop. Class is assigned 370 before the loop and increments by 1 for each loop. Once class hits 372, we break out and return class which will be 372. Result then gets returned 372.

```rust
pub fn main() {
    let mut class = 370;

    let result = loop {
        class += 1;

        if class == 372 {
            break class;
        }
    };

    println!("The class is {}", result);
}
```

```
C:\Users\louis\hello_cargo\project>cargo run
   Compiling project v0.1.0 (C:\Users\louis\hello_cargo\project)
    Finished dev [unoptimized + debuginfo] target(s) in 0.36s
     Running `target\debug\project.exe`
The class is 372
```

### 3.6 While

While loop in rust is your standard while loop as in many other languages. It keeps looping a block of code until the condition evaluates false. In the example below, the condition is while class is not 372. The code will print class and increment it by 1 until the condition equates to false.

```rust
pub fn main() {
    let mut class = 370;

    while class != 372 {
        println!("{}!", class);
        class += 1;
    }

    println!("Comparative!");
}
```

### 3.7 For

For loops in Rust works a little bit differently compared to other systems languages. Rust has a struct called Range which can act as the range you want to iterate through. You are also able to iterate through elements in a collection with the .iter() method. In our example below, we are iterating through elements of an array called classes and print out each elements.

```rust
pub fn main() {
    let classes = [372, 473, 335, 252];

    for element in classes.iter() {
        println!("the class is: {}", element);
    }
}
```

```
C:\Users\louis\hello_cargo\project>cargo run
   Compiling project v0.1.0 (C:\Users\louis\hello_cargo\project)
    Finished dev [unoptimized + debuginfo] target(s) in 0.39s
     Running `target\debug\project.exe`
the class is: 372
the class is: 473
the class is: 335
the class is: 252
```

### 3.8 Result

Result is an enum that contains the variants Ok and Err. It is used for any recoverable errors In our example, we're trying to open a file and assign it to f. f will either get passed back an instance of Ok that contains the file handle or an instance of Err that will contain more information about the type of error that has occurred. Since we have information about the type of error that has occurred, we are able to make prediction on how to handle the situation. In our example, we're trying to open a file that does not exist. When this happens, f will contain a file not found error, which our program will then try to create the file thus preventing the program from crashing. If it does however, fail to create the file, a panic! will occur and thus shut down the code. In this case the program does not crash and we get past this section of code and to our print statement that says "Past error!".

```rust
use std::fs::File;
use std::io::ErrorKind;

pub fn main() {
    let f = File::open("help_me.txt");

    let f = match f {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("help_me.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {:?}", e),
            },
            other_error => panic!("Problem opening the file: {:?}", other_error),
        },
    };
    println!("Past error!");
}
```

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.39s
     Running `target\debug\project.exe`
Past error!
```

### 3.9 Panic!

Panic! is one of two error handling types of Rust. Panic! is a macro that stops the execution of a software when it encounters an unrecoverable error. Unrecoverable errors are usually symptoms of bugs. In our example we have our program try to access a memory location outside of the array. This will cause a panic! and shut down our code. It'll notify the user of the reason and the line.

```rust
pub fn main() {
    let class = [372, 473, 337];

    class[99];
}
```

```
C:\Users\louis\hello_cargo\project>cargo run
   Compiling project v0.1.0 (C:\Users\louis\hello_cargo\project)
error: index out of bounds: the len is 3 but the index is 99
 --> panic_error.rs:4:5
  |
4 |     class[99];
  |     ^^^^^^^^^
  |
  = note: `#[deny(const_err)]` on by default
```

### 3.10 Match

Rust contains a control flow operator called match that allows you to compare a value against a a series of patterns and then execute the code based on which pattern matches. It is very similar to pattern matching in Haskell. The example below shows an enum named Class that has some defined words. The function valueinclass takes in the enum Class type and outputs a 32 bit Int by matching the Class names to a particular 32 bit number.

```
enum Class {
    Comparative,
    Automata,
    WebDev,
    Databases
}
fn value_in_class(class: Class) -> u32 {
    match class {
        Class::Comparative => 372,
        Class::Automata => 473,
        Class::WebDev => 337,
        Class::Databases => 460,
    }
}
pub fn main(){
   let com = value_in_class(Class::Comparative);
   let auto = value_in_class(Class::Automata);
   let web = value_in_class(Class::WebDev);
   let data = value_in_class(Class::Databases);
   println!("Class is {}", com);
}
```

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.36s
     Running `target\debug\project.exe`
Class is 372
```

## 4 Data Types

## 5 Subprograms

## 6 Summary

## References

[1] Rust community. Rust(programming language). Accessed: 4/6/2020.

[2] Adam Zachary Wasserman. Rust: Built to last. Accessed: 4/6/2020.

[3] Rust Developers. Rust documentation. Accessed: 4/6/2020.

[1] [2] [3]