# Introduction to Rust

@dvigneshwer

# Viki::About();

- Lives in Bengaluru, India
- Works at MuSigma Research
- Mozilla Representative in India
- Developing DeepRust Crate
- Author of Rust CookBook by Packt
- Mozilla TechSpeaker Program member

# Agenda

1. Basic Terminologies
2. Common System programming bugs
3. Why Rust?
4. Intro to Rust
5. Type System
6. Ownership and Borrowing
7. Getting started with Rust community
8. ???

# Basic Terminologies

- Low and high level language
- System programming
- Stack and heap
- Concurrency and parallelism
- Compile time and run time
- Type system
- Garbage collector
- Mutability
- Scope

# Common System Programming Errors

- Segmentation Fault
- Buffer OverFlow

# Segmentation Fault

- Dereference a null pointer

```
//declaring a null pointer
int *pointer = NULL;
//dereference a null pointer
*pointer = 1;
```

- Try to write to a portion of memory that was marked as read-only

```
// Compiler marks the constant string as read-only
char *str = "Foo";
//Leads to segfault
*str = 'b';
```

# Buffer OverFlow

- Writing and reading the past end of buffer

```
// buffer overflow
char rand_str[5];
// write past the end of buffer
strcpy(rand_str ,"Follow me @dvigneshwer in Twitter");
// read past end of the buffer
cout << "6th character " << rand_str[5] << endl;
cout << "7th character " << rand_str[6] << endl;
return 0;
```

# Sample Error Outputs

- Segmentation fault

```
viki@Vigneshwer:~/Documents/events/AI_rust_talk/cpp_samples$ ./segfault_impl.out
Pointer memory locaiton is : 0x7fffeb48afac
Pointer value is : 10
Segmentation fault (core dumped)
```

- BufferOverFlow

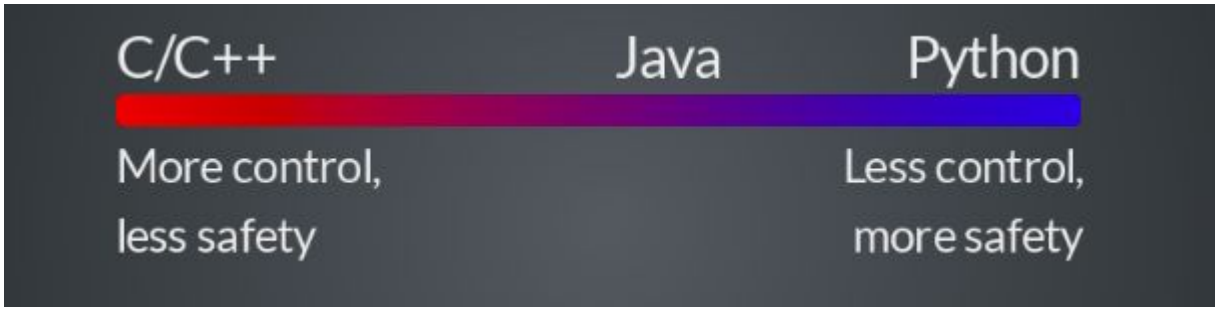
```
viki@Vigneshwer:~/Documents/events/AI_rust_talk/cpp_samples$ ./buffer_overflow.out
6th character w
7th character
*** stack smashing detected ***: ./buffer_overflow.out terminated
Aborted (core dumped)
```

# Why do we need a new system programming language?

- State or art programming language
- Solves a lot of common system programming bugs
- Cargo : Rust Package manager
- Improving your toolkit
- Self learning
- It's FUN ...

# Rust

- System programming language
- Has great control like C/C++
- Safety and expressive like python



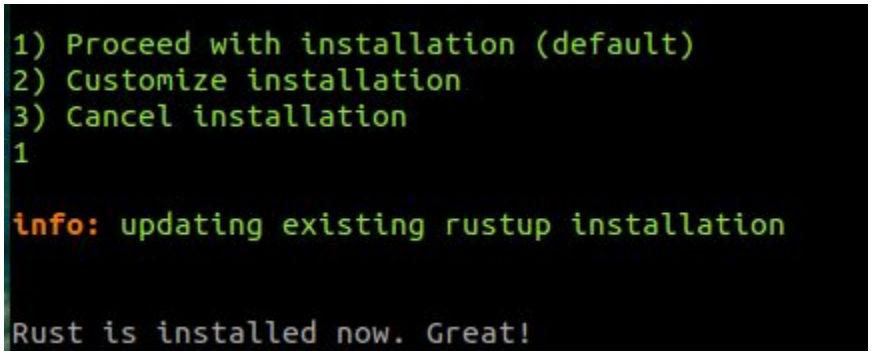
# Best things about Rust

- Strong type system
  - Reduces a lot of common bugs
- Borrowing and Ownership
  - Memory safety
  - Freedom from data races
- Zero Cost abstraction

# Installing Rust

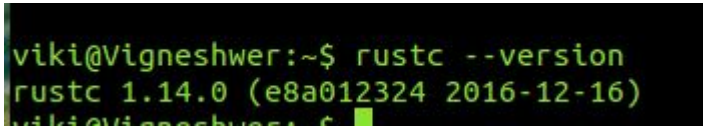
# Ubuntu / MacOS

- Open your terminal (cntrl + Alt +T)
- curl -sSf https://static.rust-lang.org/rustup.sh | sh

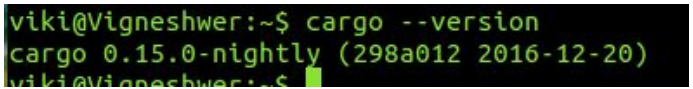


# Installing Rust

**rustc** --version




**cargo** --version




# Windows

- Go to https://win.rustup.rs/
  - This will download rustup-init.exe
- Double click and start the installation

# Type System

# Hello World

```
fn main() {
    let greet = “world”;
     println!("Hello {}!”, greet);
}
```

# A bit complex example

```
fn avg(list: &[f64]) -> f64 {
    let mut total = 0;
    for el in list{
        total += *el
    }
    total/list.len() as f64
}
```

# HLL version

```
fn avg(list: &[f64]) -> f64 {
    list.iter().sum::<f64>() / list.len() as f64
}
```

# Parallel Version (Rayon)

```
fn avg(list: &[f64]) -> f64 {
    list.par_iter().sum::<f64>() / list.len() as f64
}
```

# Fold

```
fn avg(list: &[f64]) -> f64 {
    list.par_iter().fold(0., |a,b| a + b) / list.len() as f64
}
```

# Primitive Types

# bool

```
let bool_val: bool = true;

println!("Bool value is {}", bool_val);
```

# char

```
let x_char: char = 'a';

// Printing the character

println!("x char is {}", x_char);
```

# i8/i16/i32/i64/isize

```
let num =10;

println!("Num is {}", num);

let age: i32 =40;

println!("Age is {}", age);

println!("Max i32 {}",i32::MAX);

println!("Max i32 {}",i32::MIN);
```

# Other Primitive Types

- u8/u16/u32/u64/usize
- f32/f64

# Tuples

```
// Declaring a tuple

let rand_tuple = ("Mozilla Science Lab", 2016);

let rand_tuple2 : (&str, i8) = ("Viki",4);

// tuple operations

println!(" Name : {}", rand_tuple2.0);

println!(" Lucky no : {}", rand_tuple2.1);
```

# Arrays

```
let rand_array = [1,2,3]; // Defining an array

println!("random array {:?}",rand_array );

println!("random array 1st element {}",rand_array[0] ); // indexing starts with 0

println!("random array length {}",rand_array.len() );

println!("random array {:?}",&rand_array[1..3] ); // last two elements
```

# String

let rand_string = "I love Mozilla Science <3"; **// declaring a random string**

println!("length of the string is {}",rand_string.len() ); **// printing the length of the string**

let (first,second) = rand_string.split_at(7); **// Splits in string**

let count = rand_string.chars().count(); **// Count using iterator count**

# Complex Data structures

# struct

```
// define your custom user datatype

struct Circle {

    x : f64,

    radius : f64,

}
```

# Rust “Class”

```
impl Circle {

// pub makes this function public which makes it accessible outsite the scope {}

        pub fn get_x(&self) -> f64 {

                self.x

        }

}
```

# Traits

- Interfaces
- Operator overloading
- Indicators of behaviour
- Bounds for generic
- Dynamic dispatch

# Trait Sample

```
// create a functionality for the datatypes

trait  HasArea {

        fn area(&self) -> f64;

}

// implement area for circle

impl HasArea for Circle {

        fn area(&self) -> f64 {

                3.14 * (self.r *self.r)

        }

}
```

# Ownership

In Rust, every value has an “owning scope,” and passing or returning a value means transferring ownership (“moving” it) to a new scope

```
fn make_vec() {
    let mut vec = Vec::new(); // owned by make_vec's scope
    vec.push(0);
    vec.push(1);
    // scope ends, `vec` is destroyed
}
```

# Example 1

```
fn foo{
        let v = vec![1,2,3];
        let x = v;
        println!(“{:?}”,v); // ERROR : use of moved value: “v”
}
```

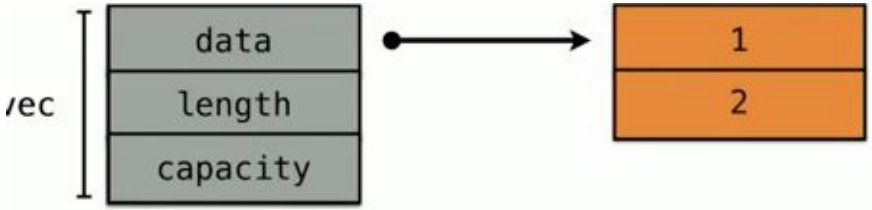
# Ownership - Ex 2

```rust
fn make_vec() -> Vec<i32> {
    let mut vec = Vec::new();
    vec.push(0);
    vec.push(1);
    vec // transfer ownership to the caller
}

fn print_vec(vec: Vec<i32>) {
    // the `vec` parameter is part of this scope, so it's owned by `print_vec`

    for i in vec.iter() {
        println!("{}", i)
    }

    // now, `vec` is deallocated
}

fn use_vec() {
    let vec = make_vec(); // take ownership of the vector
    print_vec(vec);       // pass ownership to `print_vec`
}
```

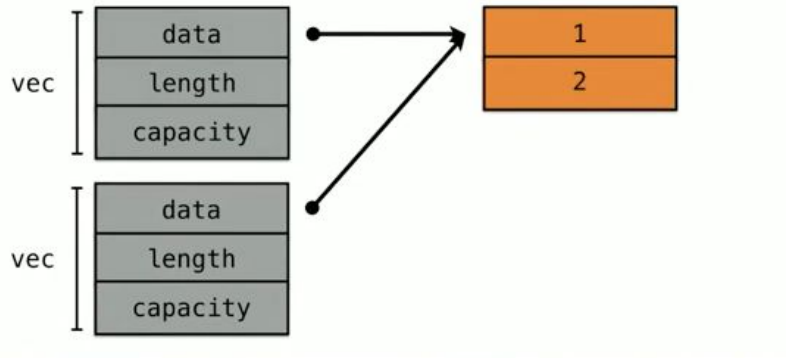
```
fn print(v : Vec<u32>) {
    println!("{:?}", v);
}
fn make_vec() {
    let v = vec![1,2,3];
    print(v);
    print(v); // ERROR : use of moved value: "v"
}
```

# Ownership

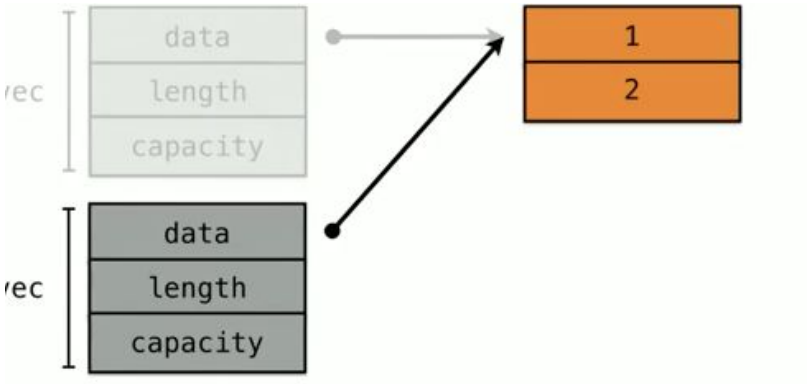
Step 1.




Step 2.

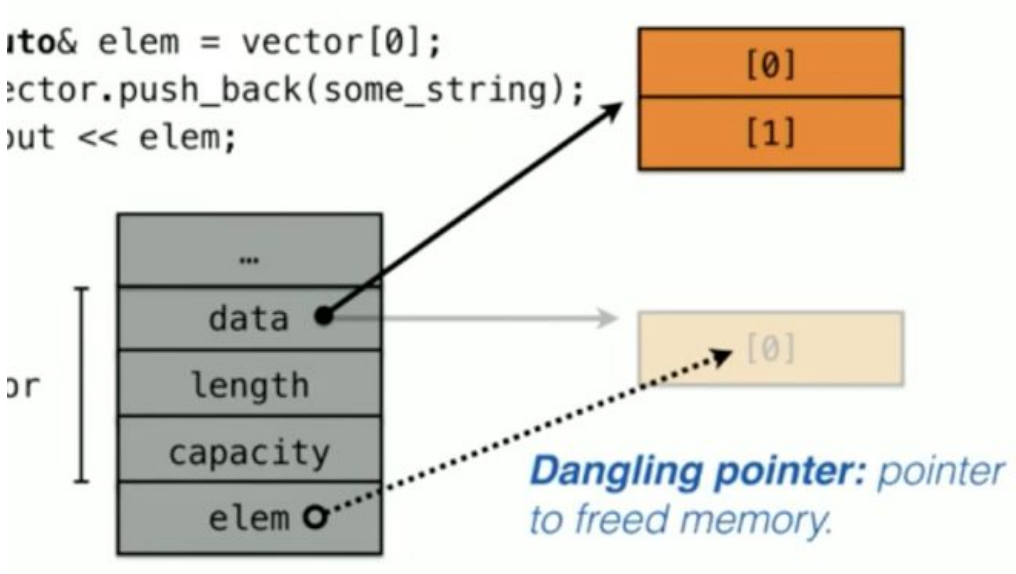



Step 3.



# Aliasing

More than one pointer to the same memory




**Ownership concepts avoids Aliasing**

# Borrowing

If you have access to a value in Rust, you can lend out that access to the functions you call

```rust
fn print_vec(vec: &Vec<i32>) {
    // the `vec` parameter is borrowed for this scope

    for i in vec.iter() {
        println!("{}", i)
    }

    // now, the borrow ends
}

fn use_vec() {
    let vec = make_vec();  // take ownership of the vector
    print_vec(&vec);       // lend access to `print_vec`
    for i in vec.iter() {  // continue using `vec`
        println!("{}", i * 2)
    }
    // vec is destroyed here
}
```

# Types of Borrowing

There is two type of borrowing in Rust, both the cases aliasing and mutation do not happen simultaneously

- Shared Borrowing (&T)
- Mutable Borrow (&mut T)

# &mut T

```
fn  add_one(v: &mut Vec<u32> ) {

        v.push(1)

}

fn foo() {

let mut v = Vec![1,2,3];

add_one(&mut v);

}
```

# Rules of Borrowing

- Mutable borrows are exclusive
- Cannot outlive the object being borrowed

# Cannot outlive the object being borrowed

```
fn foo{

let mut v = vec![1,2,3];

let borrow1 = &v;

let borrow2 = &v;

add_one(&mut v): // ERROR : cannot borrow ‘v’ as mutuable because

}                                it is also borrowed as immutable
```

# Lifetimes

```
let outer;

{

        let v = 1;

        outer = &v; // ERROR: ‘v’ doesn’t live long

}

println!(“{}”, outer);
```

# Getting started with Rust community

- Follow all the latest news at Reddit Channel
  - https://www.reddit.com/r/rust/
- Have doubts, post in
  - https://users.rust-lang.org
  - #rust IRC channel
- Want to publish a crate,
  - https://crates.io
- Follow @rustlang in twitter,
  - https://twitter.com/rustlang
- Subscribe to https://this-week-in-rust.org/ newsletter

# Getting started with Rust community

- Create your rustaceans profile,
  - Fork https://github.com/nrc/rustaceans.org
  - Create a file in data directory with <github_id>.json
    - Ex: dvigneshwer.json

```
{
    "name": "Vigneshwer Dhinakaran",
    "irc": "dvigneshwer",
    "irc_channels": ["rust"],
    "show_avatar": true,
    "email": "dvigneshwer@gmail.com",
    "discourse": "dvigneshwer",
    "reddit": "dvigneshwer",
    "twitter": "@dvigneshwer",
    "blog": "https://dvigneshwer.wordpress.com/",
    "website": "http://dvigneshwer.github.io/",
    "notes": "Innovative Data Scientist"
}
```

# Adopt Rust today !!

# References


- Segfault: http://stackoverflow.com/questions/2346806/what-is-a-segmentation-fault
- BufferOverFlow: http://stackoverflow.com/questions/574159/what-is-a-buffer-overflow-and-how-do-i-cause-one
- Rust Website: https://www.rust-lang.org/en-US/
- Community Forum: https://users.rust-lang.org/
- Rust Book: https://doc.rust-lang.org/book/
- Unraveling Rust Design: https://dvigneshwer.wordpress.com/2017/02/25/unraveling-rust-design/
- Rust Cookbook: https://www.packtpub.com/application-development/rust-cookbook

# Contribute

- https://github.com/MozillaTN/Rust
- https://github.com/dvigneshwer/deeprust
- https://github.com/dvigneshwer/Benchmarking_Rust
- https://github.com/servo

# Thank You

- Tweet at #RustIndia
- Join RustIndia Telegram group