C++ to Rust

Follow this series for more bite-sized comparisons between C++ and Rust!

✅ C++ — Immutable Reference:

```cpp
#include <iostream>
#include <string>

void PrintContact(const std::string& name) {
  std::cout << "Contact: " << name << std::endl;
}

int main() {
  std::string contact = "Alice";
  PrintContact(contact);  // No copy
  std::cout << "After: " << contact << std::endl;
  return 0;
}
```

🔍 const std::string& avoids copying while still allowing read-only access.

C++ to Rust

Follow this series for more bite-sized comparisons between C++ and Rust!

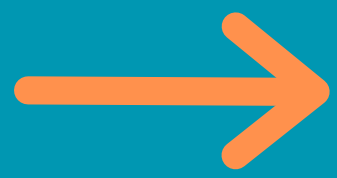
✅ C++ — Mutable Reference:

```cpp
#include <iostream>
#include <string>

void AppendLastName(std::string& name) {
  name += " Smith";
}

int main() {
  std::string contact = "Alice";
  AppendLastName(contact);  // Modifies in-place
  std::cout << "Updated: " << contact << std::endl;
  return 0;
}
```

🔧 std::string& lets us modify the original object without copying.



C++ to Rust

Follow this series for more bite-sized comparisons between C++ and Rust!

Rust — Immutable Borrowing with &T:

```rust
fn print_contact(name: &String) {
    println!("Contact: {}", name);
}

fn main() {
    let contact = String::from("Alice");
    print_contact(&contact); // Borrowed
    println!("After: {}", contact); // Still usable
}
```

📎 &String creates an immutable borrow — no copying or moving.

C++ to Rust

Follow this series for more bite-sized comparisons between C++ and Rust!

Rust — Mutable Borrowing with &mut T:

```rust
fn append_last_name(name: &mut String) {
    name.push_str(" Smith");
}

fn main() {
    let mut contact = String::from("Alice");
    append_last_name(&mut contact);
    println!("Updated: {}", contact);
}
```

🧠 Only one mutable reference is allowed at a time — enforced by the compiler!

C++ to Rust

Follow this series for more bite-sized comparisons between C++ and Rust!

✅ What to notice:

🧠 In C++:

- const T& allows read-only access without copying.
- T& gives mutable access to the original object.
- It's your responsibility to avoid issues like dangling references or aliasing bugs.

🔁 In Rust

- &T is an immutable borrow, safe and enforced by the compiler.
- &mut T is a mutable borrow, but you can only have one at a time.
- The borrow checker ensures memory safety and prevents data races at compile time.

🔍Key difference:

- In C++, reference safety is manual — the compiler doesn't protect you from misusing them.
- In Rust, reference safety is built-in — the compiler enforces borrowing rules before your code runs.