C++ to Rust

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

Variable Redeclaration vs Shadowing

C++

```cpp
#include <iostream>

auto main() -> int {
    int x = 10;
    // int x = 20;  // ❌ Error: redeclaration of 'x' in the same scope

    {
        int x = 20;  // ✅ OK: new scope, new variable
        std::cout << "Inner x: " << x << "\n";
    }

    std::cout << "Outer x: " << x << "\n";
    return 0;
}
```

C++ to Rust

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

Variable Redeclaration vs Shadowing

Rust

```rust
fn main() {
    let x = 10;
    let x = x + 5;  // ✅ shadow previous `x`
    println!("x after shadowing: {}", x);

    {
        let x = 100; // ✅ inner scope shadowing
        println!("Inner x: {}", x);
    }

    println!("Outer x: {}", x);  // outer value restored
}
```

C++ to Rust

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

✅ What to notice:

- 🧠 In C++, you can't redeclare a variable in the same scope — the compiler throws an error.
- 🔁 In Rust, you can shadow a variable by using let again in the same scope — the previous binding is overridden by a new one.
- 📦 Shadowing is often used in Rust for:
  - Type transformation
  - Immutability-to-mutability transitions
  - Rebinding after ownership moves

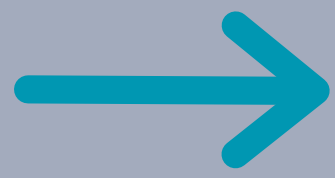
C++ to Rust

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

✅ Type transformation:

```rust
fn main() {
    let input = "42";
    let input = input.parse::<i32>().expect("Not a number");
    println!("Parsed value: {}", input);
}
```

✅ Here, input starts as a &str and is shadowed into an i32.
You don't need a new variable name like parsed_input.



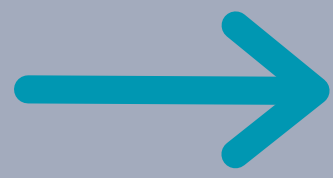
C++ to Rust

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

✅ Immutability-to-mutability transitions:

```rust
fn main() {
    let count = 0;
    let mut count = count;  // shadowing to make it mutable
    count += 1;
    println!("Count: {}", count);
}
```

✅ This is useful when you want to start with an immutable binding, but later change the value, without introducing a new variable name.



C++ to Rust

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

✅ Rebinding after ownership moves:

```rust
fn main() {
    let name = String::from("Alice");
    takes_ownership(name);

    // name is now invalid, so we shadow it with a new value
    let name = String::from("Bob");
    println!("New name: {}", name);
}

fn takes_ownership(s: String) {
    println!("Took: {}", s);
}
```

✅ After name is moved into takes_ownership, you can't use it anymore. Shadowing allows you to cleanly rebind a fresh value to name.