# WEB3CLUBS FOUNDATION LIMITED

Course Instructor: DR. Cyprian Omukhwaya Sakwa
PHONE: +254723584205 Email: cypriansakwa@gmail.com

## Foundational Mathematics for Web3 Builders

## Implemented in RUST

### Lecture 38

**July 29, 2024**

# Order of a group

- The order of the additive group $\mathbb{Z}_n$ is $n$.
- The order of the multiplicative group $\mathbb{Z}_n^*$ is $\phi(n)$ where $\phi$ is Euler's phi function defined below.

**Definition 12**

Euler phi function, denoted $\phi(n)$, is the number of positive integers less than $n$ which are relatively prime to $n$

$\mathbb{Z}_8^* = \{1, 3, 5, 7\}$

The following table gives $\phi(n)$ for $n = 1, 2, 3, \cdots, 14$. $\phi(8) = 4$

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\phi(n)$ | 1 | 1 | 2 | 2 | 4 | 2 | 6 | 4 | 6 | 4 | 10 | 4 | 12 | 6 |

In general, if $p$ is a prime number, $\phi(p) = p - 1$

$\phi(10) = \left(1-\frac{1}{2}\right) \times \left(1-\frac{1}{5}\right) 10 \Rightarrow \frac{1}{2} \times \frac{4}{5} \times 10 = 4$

If $n$ is a positive integer with prime factors $p_1, p_2, p_3, \cdots, p_k$ then $\phi(n) = (1-\frac{1}{p_1})(1-\frac{1}{p_2})(1-\frac{1}{p_3})\cdots(1-\frac{1}{p_k})n$.

**Example 59**

Find the order of $\mathbb{Z}^*_{4900}$.

**Solution**

The number of elements in $\mathbb{Z}_{4900}$ is given by $\phi(n)$.

Since $4900 = 2^2 \times 5^2 \times 7^2$, we have

$$\phi(4900) = \left(1-\frac{1}{2}\right)\left(1-\frac{1}{5}\right)\left(1-\frac{1}{7}\right)4900$$

$$= \frac{1}{2} \times \frac{4}{5} \times \frac{6}{7} \times 4900$$

$$= 1680$$

$$\begin{array}{r|l} 2 & 4900 \\ 2 & 2450 \\ 5 & 1225 \\ 5 & 245 \\ 7 & 49 \\ 7 & 7 \\ & 1 \end{array}$$

$\phi(4900) = \left(1-\frac{1}{2}\right)\left(1-\frac{1}{5}\right)\left(1-\frac{1}{7}\right)4900$

**Example 60**

Denote the Euler phi function by $\phi(n)$. Find $\phi(19)$, $\phi(840)$ and $\phi(930)$.

$\mathbb{Z}_{19}^*$, $\mathbb{Z}_{\underline{840}}^*$, $\mathbb{Z}_{930}^*$

**Solution**

Since 19 is prime, $\phi(19) = 19 - 1 = 18$

$$\phi(840) = \frac{1}{2} \times \frac{2}{3} \times \frac{4}{5} \times \frac{6}{7} \times 840$$
$$= 192$$

$$\phi(930) = \frac{1}{2} \times \frac{2}{3} \times \frac{4}{5} \times \frac{30}{31} \times 930$$
$$= 240$$

The following Rust code will calculate the $\phi(n)$.

```
fn gcd(a: u64, b: u64) -> u64 {
        if b == 0 {
                a
        } else {
                gcd(b, a % b)
        }
}

fn euler_phi(n: u64) -> u64 {
        let mut count = 0;
        for i in 1..=n {
                if gcd(n, i) == 1 {
                        count += 1;
                }
        }
        count
}

fn main() {
        let n = 4900; // You can change this to any number you want to test
 println!("Euler's␣Totient␣Function␣phi({})␣=␣{}", n, euler_phi(n));
}
```

## Understanding the Rust code

1. Greatest Common Divisor (GCD) Function

```rust
fn gcd(a: u64, b: u64) -> u64 {
        if b == 0 {
                a
        } else {
                gcd(b, a % b)
        }
}

```

- It computes the $\gcd$ of two numbers $a$ and $b$ using the Euclidean algorithm.

   a) If $b$ is $0$, the $\gcd$ is $a$.

   b) Otherwise, it recursively calls itself with $b$ and $a \bmod b$.

## Understanding the Rust code (conti...)

2. Euler's Totient Function (euler_phi).

```
fn euler_phi(n: u64) -> u64 {
        let mut count = 0;
        for i in 1..=n {
                if gcd(n, i) == 1 {
                        count += 1;
                }
        }
        count
}

```

- It computes Euler's Totient Function $\phi(n)$ for the given $n$.

   a) Initialize a counter count to $0$.
   Loop through all integers from $1$ to $n$ (inclusive).

   b) For each integer $i$, check if the $\gcd$ of $n$ and $i$ is $1$ (i.e., they are coprime).

   c) If they are coprime, increment the counter count.

   d) Return the value of count, which represents $\phi(n)$.

## Understanding the Rust code (conti...)

3. Main Function

```
fn main() {
        let n = 4900; // You can change this to any number you want to test
 println!("Euler's Totient Function phi({}) = {}", n, euler_phi(n));
}
```

- Sets a variable $n$ to the value you want to test.
- Calls the euler_phi function with $n$ and prints the result.

- To efficiently handle large numbers, we will use Rust's num-bigint crate. This crate contains the BigUint type to handle arbitrarily huge unsigned integers. In addition, we will use prime factorization to optimize the calculation of Euler's Totient function.

## Example 61

Find $\phi(4900555555555563333)$

```rust
use num_bigint::BigUint;
use num_traits::{One, Zero};
use std::str::FromStr;

fn euler_phi(mut n: BigUint) -> BigUint {
        if n.is_one() {
                return BigUint::one();
        }

        let mut result = n.clone();
        let mut p = BigUint::from(2u32);

        while &p * &p <= n {
                if &n % &p == BigUint::zero() {
                        while &n % &p == BigUint::zero() {
                                n /= &p;
                        }
                        result -= &result / &p;
                }
                p += BigUint::one();
        }

        if n > BigUint::one() {
                result -= &result / &n;
        }

```

```
        result
}

fn main() {
        let n = BigUint::from_str("4900555555555563333").unwrap();
println!("Euler's Totient Function phi({}) = {}", n, euler_phi(n.clone()));
}
```

## Understanding the Rust code

1. Euler's Totient Function
   - Initial Check: If $n$ is $1$, the function immediately returns $1$, as $\phi(1) = 1$.
   - Initialization;

     a) result is initialized to a clone of $n$. This will be modified to compute the result ($n$.clone() is used to create a copy of $n$).

     b) $p$ is initialized to $2$, the first prime number.

   - Prime Factorization Loop

     a) The loop runs while $p^2 \leq n$. This is because if $n$ has a factor larger than $\sqrt{n}$, there can only be one such factor, which must be $n$ itself if $n$ is prime.

     b) if $n \mod p = 0$, the inner loop divides $n$ by $p$ until $p$ no longer divides $n$. This effectively removes all factors of $p$ from $n$.

## Understanding the Rust code (conti...)

c) After removing all factors of $p$, result is updated as result $=$ result $\times \left(1 - \frac{1}{p}\right)$

- Final Adjustment

  a) If n is still greater than 1, then n itself is a prime factor. In this case, result is updated as result $=$ result $\times \left(1 - \frac{1}{n}\right)$

- Return

  a) The function returns the value of result, which is $\phi(n)$.

## Understanding the Rust code (conti...)

### Example 62

To find $\phi(36)$

1. Clone n: $n$ is $36$, and result is also initialized to $36$.
2. Initialize $p$ to $2$
3. Check if $p * p <= n$ :. As long as $p * p$ is less than or equal to $n$, the loop continues.
   Initially, $2 * 2 <= 36$ (true).
4. Check if $n$ is divisible by $p$.
   Check if 36 % 2 $==$ 0 (true).
5. Divide $n$ by $p$ until it is no longer divisible:

## Understanding the Rust code (conti...)

result = 36

### Example 63

p = 2

- 36 / 2 = 18
- 18 / 2 = 9
- 9 is not divisible by 2, so we exit the inner loop.

6. Update result:

   36 = result / 2

   result is 36, and p is 2 so result = result $\times \left(1 - \frac{1}{2}\right) = 18$

7 Increment p: p is incremented to 3 and repeat the above steps.

- $3 * 3 <= 9$ (true).
- 9 % 3 == 0 (true).
- 9 / 3 = 3
- 3 / 3 = 1

8 Update result:

   Result ⇒ Result = = Result / 3

   ⇒ 18 = = 18/3

   result is 18, and p is 3 so result = result $\times \left(1 - \frac{1}{3}\right) = 12$

## Understanding the Rust code (conti...)

### Example 64

8 Increment p to 4. $4 \times 4 \leq 1$

9 Check the remaining part of n

Now, $p * p > n(4 * 4 > 1)$, so we exit the loop.

n is 1, which is not greater than 1, so condition

```
if n > BigUint::one() {
        result -= &result / &n;
}
```

is not executed.

The final value of result is $12$

Thus $\phi(36) = 12$

## Understanding the Rust code (conti...)

### Example 65

To find $\phi(132)$

1. Clone n: $n$ is $132$, and result is also initialized to $132$.
2. Initialize $p$ to $2$
3. Check if $p * p <= n$ :. As long as $p * p$ is less than or equal to $n$, the loop continues.
   Initially, $2 * 2 <= 132$ (true).
4. Check if $n$ is divisible by $p$.
   Check if 132 % 2 == 0 (true).
5. Divide $n$ by $p$ until it is no longer divisible:

## Understanding the Rust code (conti...)

### Example 66

- 132 / 2 = 66
- 66 / 2 = 33
- 33 is not divisible by 2, so we exit the inner loop.

6. Update result:
   result is 132, and p is 2 so result $= 132 \times \left(1 - \frac{1}{2}\right) = 66$
7. Increment p: p is incremented to 3 and repeat the above steps.

- $3 * 3 <= 33$ (true).
- 33 % 3 == 0 (true).
- 33 / 3 = 11
- 11 is not divisible by 3, so we exit the inner loop.

8. Update result:
   result is 66, and p is 3 so result $= 66 \times \left(1 - \frac{1}{3}\right) = 44$

## Understanding the Rust code (conti...)

### Example 67

9. Increment p to 5.
10. Check the remaining part of n

    $5 * 5 > 11$), (false), increment p to 7.

    $7 * 7 > 11$), (false), increment p to 8.

11. so we exit the loop.

Now $n$ is $11$, which is greater than 1, so condition so 11 is prime

```
if n > BigUint::one() {
        result -= &result / &n;
}
```

is executed.

## Understanding the Rust code (conti...)

**Example 68**

Here, result is $44$ and $n = 11$ so final value of result is $44 \times \left(1 - \frac{1}{11}\right) = 40$

Thus $\phi(132) = 40$