1. Write a RUST program to display “hello world” message.

```
fn main() {
   println!("Helo World!!");
}
```

2. Write a RUST program to demonstrate variables, mutability and type references.

```
// Online Rust compiler to run Rust program online
// Print "Try programiz.pro" message

fn main() {
   // Immutable variable
   let x = 5;
   println!("The value of x is: {}", x);

   // Mutable variable
   let mut y = 10;
   println!("The value of y is: {}", y);
   y = 15;
   println!("The value of y after mutation is: {}", y);

   // References
   let z = 20;
   let z_ref = &z;
   println!("The value of z is: {}", z);
   println!("The value of z_ref is: {}", z_ref);

   // Mutable reference
   let mut a = 25;
```

```
    {
        let a_ref = &mut a;
        *a_ref += 5;
    }
    println!("The value of a after mutation through reference is: {}", a);
}
```

OutPut:

```
The value of x is: 5
The value of y is: 10
The value of y after mutation is: 15
The value of z is: 20
The value of z_ref is: 20
The value of a after mutation through reference is: 30
```

3. Write a program to demonstrate Rust Type Casting

```
fn main() {
    // Integer casting
    let int_value: i32 = 42;
    let float_value: f64 = int_value as f64;
    println!("The value of int_value is: {}", int_value);
    println!("The value of float_value after casting from int_value is: {}", float_value);

    // Float casting
    let another_float: f64 = 3.14159;
    let truncated_int: i32 = another_float as i32;
    println!("The value of another_float is: {}", another_float);
    println!("The value of truncated_int after casting from another_float is: {}", truncated_int);
    // Casting to smaller integer type
    let big_int: i64 = 123456;
    let small_int: i16 = big_int as i16;
    println!("The value of big_int is: {}", big_int);
    println!("The value of small_int after casting from big_int is: {}", small_int);

    // Character to integer casting
    let char_value: char = 'A';
    let char_to_int: u8 = char_value as u8;
    println!("The value of char_value is: {}", char_value);
    println!("The ASCII value of char_value after casting is: {}", char_to_int);
}
```

OutPut:

```
The value of int_value is: 42
The value of float_value after casting from int_value is: 42
The value of another_float is: 3.14159
The value of truncated_int after casting from another_float is: 3
The value of big_int is: 123456
The value of small_int after casting from big_int is: -7616
The value of char_value is: A
The ASCII value of char_value after casting is: 65
```

3. Write a RUST program to perform basic arithmetic operations on two given numbers using arithmetic operators.

```
use std::io;
```

```
fn main() {
    // Input numbers
    println!("Enter the first number:");
    let mut input1 = String::new();
    io::stdin().read_line(&mut input1).expect("Failed to read input");
    let num1: f64 = input1.trim().parse().expect("Invalid number");

    println!("Enter the second number:");
    let mut input2 = String::new();
    io::stdin().read_line(&mut input2).expect("Failed to read input");
    let num2: f64 = input2.trim().parse().expect("Invalid number");

    // Perform arithmetic operations
    let sum = num1 + num2;
    let difference = num1 - num2;
    let product = num1 * num2;
    let quotient = num1 / num2;
    let remainder = num1 % num2;

    // Output results
    println!("Sum: {} + {} = {}", num1, num2, sum);
    println!("Difference: {} - {} = {}", num1, num2, difference);
    println!("Product: {} * {} = {}", num1, num2, product);
    println!("Quotient: {} / {} = {}", num1, num2, quotient);
    println!("Remainder: {} % {} = {}", num1, num2, remainder);
}
```

Output:

```
Enter the first number:
```

```
20
Enter the second number:
30
Sum: 20 + 30 = 50
Difference: 20 - 30 = -10
Product: 20 * 30 = 600
Quotient: 20 / 30 = 0.6666666666666666
Remainder: 20 % 30 = 20
```

5. Write a RUST program to calculate student grades to a subject based on their overall score.
a. if the score is above 90, assign grade A
b. if the score is above 75, assign grade B
c. if the score is above 65, assign grade C

```
use std::io;

fn main() {
    // Input the student's score
    println!("Enter the student's score:");
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read input");
    let score: f64 = input.trim().parse().expect("Invalid score");

    // Calculate the grade
    let grade = if score > 90.0 {
        'A'
    } else if score > 75.0 {
        'B'
    } else if score > 65.0 {
        'C'
    } else {
        'D'
    };

    // Output the grade
    println!("The student's grade is: {}", grade);
}
```

Output:

```
Enter the student's score:
65
The student's grade is: D
```

6. Write a RUST program to print all prime numbers from 1 to n using for loop.

```
fn main() {
    // Input the upper limit
    println!("Enter the upper limit (n):");
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read input");
    let n: u32 = input.trim().parse().expect("Invalid number");

    println!("Prime numbers from 1 to {}:", n);
    for num in 2..=n {
        let mut is_prime = true;
        for i in 2..=((num as f64).sqrt() as u32) {
            if num % i == 0 {
                is_prime = false;
                break;
            }
        }
        if is_prime {
            println!("{}", num);
        }
    }
}
```

Output:

```
Enter the upper limit (n):
12
Prime numbers from 1 to 12:
2
3
5
7
11
```

7. Write a program to demonstrate the usage of functions in RUST to find sum of two
numbers. Pass two values as parameters.

```
use std::io;

fn main() {
    // Input the first number
    println!("Enter the first number:");
    let mut input1 = String::new();
    io::stdin().read_line(&mut input1).expect("Failed to read input");
    let num1: i32 = input1.trim().parse().expect("Invalid number");

    // Input the second number
    println!("Enter the second number:");
    let mut input2 = String::new();
    io::stdin().read_line(&mut input2).expect("Failed to read input");
    let num2: i32 = input2.trim().parse().expect("Invalid number");

    // Call the sum function and print the result
    let result = sum(num1, num2);
    println!("The sum of {} and {} is: {}", num1, num2, result);
}

// Function to find the sum of two numbers
fn sum(a: i32, b: i32) -> i32 {
    a + b
}
```

Output:

```
Enter the first number:
20
Enter the second number:
30
The sum of 20 and 30 is: 50
```

8. Write a program to create a vector of strings and access the elements. Display all elements in sorted order.

```
use std::io;

fn main() {
    let mut strings = Vec::new();
    let mut input = String::new();
    println!("Enter strings (type 'done' to finish):");

    // Read strings from user input
    loop {
        input.clear(); // Clear the input string for new input
        io::stdin().read_line(&mut input).expect("Failed to read input");
        let trimmed_input = input.trim();

        if trimmed_input.eq("done") {
            break; // Exit the loop if user types 'done'
        }

        strings.push(trimmed_input.to_string()); // Add the input string to the vector
    }

    // Sort the vector
    strings.sort();

    // Display the sorted strings
    println!("Sorted strings:");
    for s in &strings {
        println!("{}", s);
    }
}
```

OutPut:

```
Enter strings (type 'done' to finish):
RCR
PAVAN
CHANDU
ANIL
```

```
done

Sorted strings:

ANIL

CHANDU

PAVAN

RCR
```

9. Write a RUST program to demonstrate different ways of creating iterators.

```
fn main() {
   // 1. Creating an iterator using a range
   let range_iter = 1..=5; // Inclusive range from 1 to 5
   println!("Range iterator:");
   for number in range_iter {
      println!("{}", number);
   }

   // 2. Creating an iterator from a vector
   let vec = vec![10, 20, 30, 40, 50];
   let vec_iter = vec.iter();
   println!("\nVector iterator:");
   for value in vec_iter {
      println!("{}", value);
   }

   // 3. Creating an iterator using the `into_iter` method
   let vec_into_iter = vec![100, 200, 300];
   println!("\nInto iterator:");
   for value in vec_into_iter {
      println!("{}", value);
   } // Note: vec_into_iter is consumed here

   // 4. Creating an iterator with the `map` method
   let squares: Vec<i32> = (1..=5).map(|x| x * x).collect();
   println!("\nSquares using map:");
   for square in squares.iter() {
      println!("{}", square);
   }

   // 5. Creating an iterator with the `filter` method
   let evens: Vec<i32> = (1..=10).filter(|&x| x % 2 == 0).collect();
   println!("\nEven numbers using filter:");
   for even in evens.iter() {
      println!("{}", even);
   }
}
```

Output:

```
Range iterator:
1
2
3
4
5

Vector iterator:
10
20
30
40
50

Into iterator:
100
200
300

Squares using map:
1
4
9
16
25

Even numbers using filter:
2
4
6
8
10
```

10. Write a RUST program to perform all strings operations like creation of string,slicing of stirng etc.

```
fn main() {
    // 1. Creating a string
    let mut greeting = String::from("Hello");
    println!("Original string: {}", greeting);

    // 2. Concatenating strings
    let world = String::from(", world!");
    greeting.push_str(&world);
    println!("Concatenated string: {}", greeting);

    // 3. Slicing a string
    let slice = &greeting[0..5]; // Slice the first 5 characters
    println!("Sliced string (first 5 characters): {}", slice);

    // 4. Length of the string
    println!("Length of the string: {}", greeting.len());

    // 5. Iterating over characters
    println!("Iterating over characters:");
    for c in greeting.chars() {
        println!("{}", c);
    }

    // 6. Checking if a string contains a substring
    let contains = greeting.contains("world");
    println!("Contains 'world': {}", contains);

    // 7. Replacing part of the string
    let replaced = greeting.replace("world", "Rust");
    println!("String after replacement: {}", replaced);

    // 8. Splitting a string
    println!("Splitting the string:");
    for word in greeting.split_whitespace() {
        println!("{}", word);
    }

    // 9. Converting to uppercase
    let upper = greeting.to_uppercase();
    println!("Uppercase string: {}", upper);
}
```

Output:

```
Original string: Hello
Concatenated string: Hello, world!
Sliced string (first 5 characters): Hello
Length of the string: 13
Iterating over characters:
H
e
l
l
o
,
 
w
```

```
o
r
l
d
!
Contains 'world': true
String after replacement: Hello, Rust!
Splitting the string:
Hello,
world!
Uppercase string: HELLO, WORLD!
```

11. Write a RUST program to demonstrate recoverable errors using panic, except .

```
use std::fs::File;
use std::io::{self, Read};

fn main() {
    // Recoverable error handling with Result
    let result = read_file("example.txt");

    match result {
        Ok(content) => println!("File content:\n{}", content),
        Err(e) => println!("Failed to read file: {}", e),
    }

    // Unrecoverable error using panic!
    let value = get_value(0);
    println!("Value: {}", value);
}

// Function to read a file and return its content as a String
fn read_file(filename: &str) -> Result<String, io::Error> {
    let mut file = File::open(filename).map_err(|e| {
        println!("Error opening file: {}", e);
        e
    })?;

    let mut contents = String::new();
    file.read_to_string(&mut contents)?;

    Ok(contents)
}

// Function that demonstrates an unrecoverable error using panic!
fn get_value(num: i32) -> i32 {
    if num == 0 {
        panic!("Attempted to get value for zero! This is an unrecoverable error.");
    }
    num * 2
}
```

Output:

```
File content: Hello, world! Value: 10
```

13. Write a RUST program to demonstrate data movement and ownership rules in Rust

```
fn main() {
    let s1 = String::from("hello");
    let s2 = s1;

    // The following line would cause a compile-time error because s1 has been
moved to s2
    // println!("{}", s1);

    println!("{}", s2);

    let x = 5;
    let y = x;

    // This works fine because integers are Copy types, so x was copied into y, not
moved
    println!("x: {}, y: {}", x, y);

    let s3 = String::from("world");
    takes_ownership(s3);

    // The following line would cause a compile-time error because s3 has been
moved to the function
    // println!("{}", s3);

    let z = 10;
    makes_copy(z);

    // This works fine because integers are Copy types, so z was copied into the
function
    println!("z: {}", z);
}

fn takes_ownership(some_string: String) {
    println!("{}", some_string);
}

fn makes_copy(some_integer: i32) {
    println!("{}", some_integer);
}
```

Output:

```
hello
x: 5, y: 5
world
10
z: 10
```

14. Write a RUST program to demonstrate ownership in functions.

```
fn main() {
    let s1 = String::from("hello");
    let s2 = takes_ownership(s1);

    // The following line would cause a compile-time error because s1's ownership has been moved to the function
    // println!("{}", s1);

    println!("{}", s2);

    let x = 5;
    let y = makes_copy(x);

    // This works fine because integers are Copy types, so x was copied into the function
    println!("x: {}, y: {}", x, y);
}

fn takes_ownership(some_string: String) -> String {
    println!("Inside takes_ownership: {}", some_string);
    some_string  // Returning the ownership
}

fn makes_copy(some_integer: i32) -> i32 {
    println!("Inside makes_copy: {}", some_integer);
    some_integer  // Copy types are returned by copying
}
```

Output:

```
Inside takes_ownership: hello
hello
Inside makes_copy: 5
x: 5, y: 5
```

15. Demonstrate Building and Running Project with Cargo in Rust

Step-by-Step Guide

1. **Install Rust and Cargo**: Ensure you have Rust and Cargo installed. You can install them using rustup:

   sh
   Copy code

   ```
   curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
   ```

2. **Create a New Project**: Use Cargo to create a new project. Open a terminal and run:

   sh
   Copy code

   ```
   cargo new hello_world
   cd hello_world
   ```

   This creates a new directory called hello_world with the following structure:

   css
   Copy code

   ```
   hello_world
   ├── Cargo.toml
   └── src
       └── main.rs
   ```

3. **Project Structure**:
   - Cargo.toml: This is the manifest file for the project, where dependencies and metadata are defined.
   - src/main.rs: This is the main source file for the project. It contains the entry point of the program.
4. **Write the Code**: Open src/main.rs and you should see the default "Hello, world!" code:

   rust
   Copy code

   ```
   fn main() {
       println!("Hello, world!");
   }
   ```

5. **Build the Project**: To compile the project, run:

   sh
   Copy code

   ```
   cargo build
   ```

   This command compiles the code and places the executable in the target/debug directory.

6. **Run the Project**: To run the compiled executable, use:

   sh
   Copy code

   ```
   cargo run
   ```

   This command compiles the code (if necessary) and runs the resulting executable. You should see:

Copy code

```
Hello, world!
```

7. **Release Build**: To create an optimized release build, run:

   sh
   Copy code

   ```
   cargo build --release
   ```

   This places the optimized executable in the target/release directory.

15. Write a program to demonstrate Defining, Implementing and Using a Trait in Rust

Step-by-Step Guide

1. **Define a Trait**: We'll start by defining a trait called Summary that has a method summarize.
2. **Implement the Trait for a Struct**: Next, we'll create a struct Article and implement the Summary trait for it.
3. **Use the Trait**: Finally, we'll create an instance of Article and call the summarize method.

Full Example

rust

```
// Define the Summary trait
trait Summary {
    fn summarize(&self) -> String;
}

// Define a struct Article
struct Article {
    headline: String,
    author: String,
    content: String,
}

// Implement the Summary trait for Article
impl Summary for Article {
    fn summarize(&self) -> String {
        format!("{}, by {}: {}", self.headline, self.author, self.content)
    }
}

// Function that takes a reference to a type that implements Summary
fn notify(item: &impl Summary) {
    println!("Breaking news! {}", item.summarize());
}

fn main() {
```

```
    // Create an instance of Article
    let article = Article {
        headline: String::from("Rust Language Gains Popularity"),
        author: String::from("Jane Doe"),
        content: String::from("Rust is becoming more popular due to its memory safety and performance."),
    };

    // Call the summarize method
    println!("Article summary: {}", article.summarize());

    // Use the notify function
    notify(&article);
}
```

Explanation

1. **Defining the Trait**:

   rust
   ```
   trait Summary {
       fn summarize(&self) -> String;
   }
   ```

   The `Summary` trait defines a method `summarize` that takes an immutable reference to `self` and returns a `String`.

2. **Implementing the Trait for a Struct**:

   rust
   ```
   struct Article {
       headline: String,
       author: String,
       content: String,
   }

   impl Summary for Article {
       fn summarize(&self) -> String {
           format!("{}, by {}: {}", self.headline, self.author, self.content)
       }
   }
   ```

   The `Article` struct has three fields: `headline`, `author`, and `content`. We implement the `Summary` trait for `Article`, providing a concrete implementation for the `summarize` method.

3. **Using the Trait**:

   rust
   ```
   fn notify(item: &impl Summary) {
       println!("Breaking news! {}", item.summarize());
   }

   fn main() {
       let article = Article {
           headline: String::from("Rust Language Gains Popularity"),
   ```

```
        author: String::from("Jane Doe"),
        content: String::from("Rust is becoming more popular due to its memory safety and
performance."),
    };

    println!("Article summary: {}", article.summarize());

    notify(&article);
}
```

- notify is a function that takes a reference to any type that implements the Summary trait. It calls the summarize method on the passed item.
- In main, we create an instance of Article and call summarize on it.
- We also call notify, passing a reference to the article.

17. Write a RUST program to demonstrate about pattern matching.

```
enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter,
}

fn value_in_cents(coin: Coin) -> u8 {
    match coin {
        Coin::Penny => {
            println!("Lucky penny!");
            1
        }
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter => 25,
    }
}

fn main() {
    // Match with enums
    let coin = Coin::Penny;
    let value = value_in_cents(coin);
    println!("Value in cents: {}", value);

    // Match with Option<T>
    let some_number = Some(5);
    match some_number {
        Some(5) => println!("The number is five!"),
        Some(x) => println!("Matched, value: {}", x),
        None => println!("No value"),
    }
    // Matching with ranges and multiple patterns
    let dice_roll = 9;
    match dice_roll {
        1 | 2 => println!("Rolled a one or two"),
        3..=7 => println!("Rolled a three through seven"),
        _ => println!("Rolled something else"),
    }

    // Destructuring and matching with structs
    struct Point {
        x: i32,
        y: i32,
    }

    let p = Point { x: 0, y: 7 };
    match p {
        Point { x: 0, y } => println!("On the y-axis at {}", y),
        Point { x, y: 0 } => println!("On the x-axis at {}", x),
        Point { x, y } => println!("On neither axis: ({}, {})", x, y),
    }

    // Matching with guards
    let num = Some(4);
    match num {
        Some(x) if x < 5 => println!("Less than five: {}", x),
        Some(x) => println!("{}", x),
        None => (),
    }

    // Using if let for simpler matching
    let config_max = Some(3u8);
    if let Some(max) = config_max {
        println!("The maximum is configured to be {}", max);
    }

    // Using while let for looping with patterns
    let mut stack = vec![1, 2, 3];
    while let Some(top) = stack.pop() {
        println!("Popped value: {}", top);
    }
}
```

Output:

```
Lucky penny!
Value in cents: 1
The number is five!
Rolled something else
On the y-axis at 7
```

```
Less than five: 4
The maximum is configured to be 3
Popped value: 3
Popped value: 2
Popped value: 1
```

18. Implement Generic struct and Generic Functions in Rust.

```
// Define a generic struct Point with two fields, x and y, of the same generic type T
struct Point<T> {
    x: T,
    y: T,
}

// Implement methods for the generic struct Point
impl<T> Point<T> {
    fn new(x: T, y: T) -> Self {
        Point { x, y }
    }
}

// Implement methods specifically for Point<f32>
impl Point<f32> {
    fn distance_from_origin(&self) -> f32 {
        (self.x.powi(2) + self.y.powi(2)).sqrt()
    }
}

fn main() {
    // Create an instance of Point with integer coordinates
    let int_point = Point::new(5, 10);
    println!("int_point: ({}, {})", int_point.x, int_point.y);

    // Create an instance of Point with float coordinates
    let float_point = Point::new(1.0, 4.0);
    println!("float_point: ({}, {})", float_point.x, float_point.y);

    // Use the distance_from_origin method for a Point<f32>
    let float_point = Point::new(3.0, 4.0);
    println!("Distance from origin: {}", float_point.distance_from_origin());
}
```

Output:

```
int_point: (5, 10)
float_point: (1, 4)
Distance from origin: 5
```

19. Write a RUST program for performing following FILE operations:
a) Opening a file

b) Reading from a file
c) Writing to a file
d) Removing a file
e) Appending to a file

```
use std::fs::{self, File, OpenOptions};
use std::io::{self, Read, Write};

fn main() -> io::Result<()> {
    let file_path = "example.txt";

    // a) Opening a file (creating if it doesn't exist)
    let mut file = File::create(file_path)?;
    println!("File created successfully!");

    // b) Writing to a file
    file.write_all(b"Hello, Rust!")?;
    println!("Written to file!");

    // c) Reading from a file
    let mut file = File::open(file_path)?;
    let mut contents = String::new();
    file.read_to_string(&mut contents)?;
    println!("File contents: {}", contents);

    // d) Appending to a file
    let mut file = OpenOptions::new()
        .append(true)
        .open(file_path)?;
    file.write_all(b"\nWelcome to file operations in Rust!")?;
    println!("Appended to file!");

    // Reading the appended file
    let mut file = File::open(file_path)?;
    let mut new_contents = String::new();
    file.read_to_string(&mut new_contents)?;
    println!("Updated file contents: {}", new_contents);

    // e) Removing a file
    fs::remove_file(file_path)?;
    println!("File removed successfully!");

    Ok(())
}
```

Output:

20. Build a multithreaded web server using RUST.

Step-by-Step Guide

Step1 :

Set Up the Project: Create a new Rust project using Cargo.

```
cargo new multithreaded_web_server
cd multithreaded_web_server
```

Step2:

Add Necessary Code: Replace the contents of src/main.rs with the following code:

```
use std::fs;
use std::io::prelude::*;
use std::net::{TcpListener, TcpStream};
use std::thread;
use std::sync::mpsc;
use std::sync::Arc;
use std::sync::Mutex;

fn main() {
    // Bind the TcpListener to the local address
    let listener = TcpListener::bind("127.0.0.1:7878").unwrap();
    let pool = ThreadPool::new(4); // Create a thread pool with 4 threads

    for stream in listener.incoming() {
        let stream = stream.unwrap();

        pool.execute(|| {
            handle_connection(stream);
        });
    }
}

fn handle_connection(mut stream: TcpStream) {
    let mut buffer = [0; 1024];
    stream.read(&mut buffer).unwrap();

    let get = b"GET / HTTP/1.1\r\n";

    let (status_line, filename) = if buffer.starts_with(get) {
        ("HTTP/1.1 200 OK", "hello.html")
    } else {
        ("HTTP/1.1 404 NOT FOUND", "404.html")
    };

    let contents = fs::read_to_string(filename).unwrap();

    let response = format!(
        "{}\r\nContent-Length: {}\r\n\r\n{}",
        status_line,
        contents.len(),
        contents
    );

    stream.write(response.as_bytes()).unwrap();
    stream.flush().unwrap();
}

struct ThreadPool {
    workers: Vec<Worker>,
    sender: mpsc::Sender<Job>,
}

type Job = Box<dyn FnOnce() + Send + 'static>;

impl ThreadPool {
    fn new(size: usize) -> ThreadPool {
        assert!(size > 0);

        let (sender, receiver) = mpsc::channel();

        let receiver = Arc::new(Mutex::new(receiver));

        let mut workers = Vec::with_capacity(size);

        for id in 0..size {
            workers.push(Worker::new(id, Arc::clone(&receiver)));
        }

        ThreadPool { workers, sender }
    }

    fn execute<F>(&self, f: F)
    where
        F: FnOnce() + Send + 'static,
    {
        let job = Box::new(f);

        self.sender.send(job).unwrap();
    }
}

struct Worker {
    id: usize,
    thread: Option<thread::JoinHandle<()>>,
}

impl Worker {
    fn new(id: usize, receiver: Arc<Mutex<mpsc::Receiver<Job>>>) -> Worker {
        let thread = thread::spawn(move || loop {
            let job = receiver.lock().unwrap().recv().unwrap();

            println!("Worker {} got a job; executing.", id);

            job();
        });

        Worker {
            id,
            thread: Some(thread),
        }
    }
}
```

Step3:
Create HTML Files:
Create two HTML files, hello.html and 404.html, in the root of your project.

hello.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello</title>
</head>
<body>
    <h1>Hello, world!</h1>
</body>
</html>
```

404.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Not Found</title>
</head>
<body>
    <h1>404 - Not Found</h1>
</body>
</html>
```

Step4:
Explanation:

TcpListener:

rust

```
let listener = TcpListener::bind("127.0.0.1:7878").unwrap();
```

ThreadPool:

```
struct ThreadPool {
    workers: Vec<Worker>,
    sender: mpsc::Sender<Job>,
}

type Job = Box<dyn FnOnce() + Send + 'static>;
```

Worker:

```
struct Worker {
    id: usize,
    thread: Option<thread::JoinHandle<()>>,
}
```

Handling Connections:

```
fn handle_connection(mut stream: TcpStream) {
    let mut buffer = [0; 1024];
    stream.read(&mut buffer).unwrap();

    let get = b"GET / HTTP/1.1\r\n";

    let (status_line, filename) = if buffer.starts_with(get) {
        ("HTTP/1.1 200 OK", "hello.html")
    } else {
        ("HTTP/1.1 404 NOT FOUND", "404.html")
    };

    let contents = fs::read_to_string(filename).unwrap();

    let response = format!(
        "{}\r\nContent-Length: {}\r\n\r\n{}",
        status_line,
        contents.len(),
        contents
    );

    stream.write(response.as_bytes()).unwrap();
    stream.flush().unwrap();
}
```

Step 5.

Running the Server:

```
cargo build
cargo run
```

Step 6:
Testing the Server:

Open your web browser and navigate to http://127.0.0.1:7878. You should see the hello.html content. Navigate to any other URL, e.g., http://127.0.0.1:7878/unknown, and you should see the 404.html content.