ECE 522 Assignment 3

Zhaoyi Wang 1689747

Question 1

For `calculator.rs`:

```rust
use std::ops::*;

pub fn add(x: f64, y: f64) -> f64 {
    x + y
}

pub fn subtract(x: f64, y: f64) -> f64 {
    x - y
}

pub fn multiply<T: Mul>(x: T, y: T) -> T::Output {
    x * y
}

pub fn divide<T: Div>(x: T, y: T) -> T::Output {
    x / y
}

pub fn get_squre_root(x: f64) -> f64 {
    if x < 0.0 {
        panic!("Negative numbers don't have real square roots!");
    }
    x.sqrt()
}

pub fn get_roots(a: f64, b: f64, c: f64) -> (f64, f64) {
    // quadratic equations: ax^2+bx+c=0
    // (-b + sqrt(b^2-4ac))/(2a) or (-b - sqrt(b^2-4ac))/(2a)
    let delta = b * b - 4.0 * a * c;
    if delta >= 0.0 {
        let root_one = (-b + delta.sqrt()) / (2.0 * a);
        let root_two = (-b - delta.sqrt()) / (2.0 * a);
        (root_one, root_two)
    } else {
        panic!("There is no root for this equation!");
        println!("There is no root for this equation!");
    }
}

```

For `question1.rs`:

```rust
use rand::prelude::*;

// Note this useful idiom: importing names from outer (for mod tests) scope.
use super::*;

#[test]
pub fn basic_add() {
    assert_eq!(calculator::add(1.0, 2.0), 3.0);
}
#[test]
pub fn add_negative_number() {
     assert_eq!(calculator::add(-1.0, 2.0), 1.0);
}
#[test]
pub fn add_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() { // random bool
    let x: f64 = rng.gen(); // random number in range [0, 1)
    let y: f64 = rng.gen();
    assert_eq!(calculator::add(x, y), x+y);
    }
}

#[test]
pub fn basic_subtract() {
    assert_eq!(calculator::subtract(4.0, 2.0), 2.0);
}
#[test]
pub fn subtract_negative_number() {
    assert_eq!(calculator::subtract(-3.0, 2.0), -5.0);
}
#[test]
pub fn subtract_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() { // random bool
    let x: f64 = rng.gen(); // random number in range [0, 1)
    let y: f64 = rng.gen();
    assert_eq!(calculator::subtract(x, y), x-y);
    }
}
```

For `question2.rs`:

```rust
use super::*;
use rand::prelude::*;

#[test]
pub fn basic_multiply() {
    let x = 4;
    let y = 2;
    assert_eq!(calculator::multiply(x, y), 8);
}

#[test]
pub fn multiply_negative_number() {
    let x = -4;
    let y = -2;
    assert_eq!(calculator::multiply(x, y), 8)
}

```

```rust
#[test]
pub fn multiply_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() {
        // random bool
        let x: f64 = rng.gen(); // random number in range [0, 1)
        let y: f64 = rng.gen();
        assert_eq!(calculator::multiply(x, y), x * y);
    }
}

#[test]
pub fn basic_divide() {
    let x = 2;
    let y = 2;
    assert_eq!(calculator::divide(x, y), 1);
}

#[test]
pub fn divide_negative_number() {
    let x = -2;
    let y = -2;
    assert_eq!(calculator::divide(x, y), 1);
}
#[test]
pub fn divide_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() {
        // random bool
        let x: f64 = rng.gen(); // random number in range [0, 1)
        let y: f64 = rng.gen();
        assert_eq!(calculator::divide(x, y), x / y);
    }
}
```

For `question3.rs`:

```rust
use super::*;
use rand::*;

#[test]
pub fn test_random_positive_square_root() {
    // let rng = rand::thread_rng().gen::<f64>();
    let rng = rand::thread_rng().gen_range(0.0..100.0);
    assert_eq!(calculator::get_squre_root(rng), rng.sqrt());
}
#[test]
#[should_panic]
pub fn test_random_negitive_square_root() {
    let rng = rand::thread_rng().gen_range(-100.0..0.0);
    assert_eq!(calculator::get_squre_root(rng), (rng).sqrt());
}
#[test]
pub fn test_square_root_of_zero() {
    let x = 0.0;
    assert_eq!(calculator::get_squre_root(x), 0.0);
}

#[test]
pub fn test_square_root_of_one() {
```

```
    let x = 1.0;
    assert_eq!(calculator::get_squre_root(x), 1.0);
}
```

For question4.rs:

```
use super::*;
use rand::*;

#[test]
pub fn test_basic_roots() {
    // y = x^2 + 6x - 7
    let a = 1.0;
    let b = 6.0;
    let c = -7.0;
    assert_eq!(calculator::get_roots(a, b, c), (1.0, -7.0));
}
#[test]
pub fn test_single_root() {
    // y = x^2
    let a = 1.0;
    let b = 0.0;
    let c = 0.0;
    assert_eq!(calculator::get_roots(a, b, c), (0.0, 0.0));
}
#[test]
pub fn test_random_solvable_quadratic() {
    let a = rand::thread_rng().gen_range(0.0..2.0);
    let b = rand::thread_rng().gen_range(20.0..100.0);
    let c = rand::thread_rng().gen_range(0.0..10.0);
    let res = calculator::get_roots(a,b,c);
    assert_eq!((a*res.0*res.0+b*res.0+c).trunc(),0.0);
}
#[test]
#[should_panic]
pub fn test_random_non_solvable_quadratic() {
    let a = rand::thread_rng().gen_range(2.0..10.0);
    let b = rand::thread_rng().gen_range(0.0..5.0);
    let c = rand::thread_rng().gen_range(10.0..20.0);
    assert_eq!(calculator::get_roots(a, b, c), (0.0, 0.0));
}
```

For the output:

```
coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3$ cargo test
   Compiling UTApp v0.1.0 (/home/coder/personalProj/rusttest/Assign3)

running 20 tests
test question1::add_negative_number ... ok
test question1::add_random_numbers ... ok
test question1::basic_add ... ok
test question1::basic_subtract ... ok
test question1::subtract_negative_number ... ok
test question1::subtract_random_numbers ... ok
test question2::basic_divide ... ok
test question2::basic_multiply ... ok
test question2::divide_negative_number ... ok
test question2::divide_random_numbers ... ok
test question2::multiply_negative_number ... ok
```

```
test question2::multiply_random_numbers ... ok
test question3::test_random_negitive_square_root ... ok
test question3::test_random_positive_square_root ... ok
test question3::test_square_root_of_one ... ok
test question3::test_square_root_of_zero ... ok
test question4::test_basic_roots ... ok
test question4::test_random_non_solvable_quadratic - should panic ... ok
test question4::test_random_solvable_quadratic ... ok
test question4::test_single_root ... ok

test result: ok. 20 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in
0.02s
```

Question 2

For `main.rs`:

```rust
mod question;
mod player;

#[cfg(test)] #[macro_use] extern crate hamcrest;

fn main() {}
```

For `player.rs`:

```rust
pub struct Player {
    pub id: i32,
    pub first_name: String,
    pub last_name: String,
}
```

For `question.rs`:

```rust
use super::*;
use hamcrest::*;

#[test]
fn name_is_String() {
    let player_one = player::Player {
        id: 001,
        first_name: String::from("Tim"),
        last_name: String::from("White"),
    };

    let player_two = player::Player {
        id: 002,
        first_name: String::from("Mike"),
        last_name: String::from("Thompson"),
    };

    assert_that!(player_one.first_name, is(type_of::<String>()));
    assert_that!(player_one.last_name, is(type_of::<String>()));
}
#[test]
```

```rust
fn all_the_same(){
    let player_one = player::Player {
        id: 001,
        first_name: String::from("Tim"),
        last_name: String::from("White"),
    };

    let player_two = player::Player {
        id: 001,
        first_name: String::from("Tim"),
        last_name: String::from("White"),
    };

    assert_that!(player_one.id,is(equal_to(player_two.id)));
    assert_that!(player_one.first_name,is(equal_to(player_two.first_name)));
    assert_that!(player_one.last_name,is(equal_to(player_two.last_name)));
}
```

For the output:

```
coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T2$ cargo test
   Compiling A3T2 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T2)

running 2 tests
test question::all_the_same ... ok
test question::name_is_String ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s
```

Question 3

For `question1.rs`:

```rust
use hamcrest::prelude::*;
use rand::prelude::*;

// Note this useful idiom: importing names from outer (for mod tests) scope.
use super::*;

#[test]
pub fn basic_add() {
    assert_that!(calculator::add(1.0, 2.0), is(equal_to(3.0)));
}
#[test]
pub fn add_negative_number() {
    assert_that!(calculator::add(-1.0, 2.0), is(equal_to(1.0)));
}
#[test]
pub fn add_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() {
        // random bool
        let x: f64 = rng.gen(); // random number in range [0, 1)
        let y: f64 = rng.gen();
        assert_that!(calculator::add(x, y), is(equal_to(x + y)));
    }
```

```
}

#[test]
pub fn basic_subtract() {
    assert_that!(calculator::subtract(4.0, 2.0), is(equal_to(2.0)));
}
#[test]
pub fn subtract_negative_number() {
    assert_that!(calculator::subtract(-3.0, 2.0), is(equal_to(-5.0)));
}
#[test]
pub fn subtract_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() {
        // random bool
        let x: f64 = rng.gen(); // random number in range [0, 1)
        let y: f64 = rng.gen();
        assert_that!(calculator::subtract(x, y), is(equal_to(x - y)));
    }
}
```

For `question2.rs`:

```
use super::*;
use hamcrest::prelude::*;
use rand::prelude::*;

#[test]
pub fn basic_multiply() {
    let x = 4;
    let y = 2;
    assert_that!(calculator::multiply(x, y), is(equal_to(8)));
}

#[test]
pub fn multiply_negative_number() {
    let x = -4;
    let y = -2;
    assert_that!(calculator::multiply(x, y), is(equal_to(8)));
}

#[test]
pub fn multiply_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() {
        // random bool
        let x: f64 = rng.gen(); // random number in range [0, 1)
        let y: f64 = rng.gen();
        assert_that!(calculator::multiply(x, y), is(equal_to(x * y)));
    }
}

#[test]
pub fn basic_divide() {
    let x = 2;
    let y = 2;
    assert_that!(calculator::divide(x, y), is(equal_to(1)));
}

#[test]
```

```rust
pub fn divide_negative_number() {
    let x = -2;
    let y = -2;
    assert_that!(calculator::divide(x, y), is(equal_to(1)));
}
#[test]
pub fn divide_random_numbers() {
    let mut rng = thread_rng();
    if rng.gen() {
        // random bool
        let x: f64 = rng.gen(); // random number in range [0, 1)
        let y: f64 = rng.gen();
        assert_that!(calculator::divide(x, y), is(equal_to(x / y)));
    }
}
```

For `question3.rs`:

```rust
use super::*;
use rand::*;
use hamcrest::prelude::*;

#[test]
pub fn test_random_positive_square_root() {
    // let rng = rand::thread_rng().gen::<f64>();
    let rng = rand::thread_rng().gen_range(0.0..100.0);
    assert_that!(calculator::get_squre_root(rng), equal_to((rng as f64).sqrt()));
}
#[test]
#[should_panic]
pub fn test_random_negitive_square_root() {
    let rng = rand::thread_rng().gen_range(-100.0..0.0);
    assert_that!(calculator::get_squre_root(rng), equal_to((rng as f64).sqrt()));
}
#[test]
pub fn test_square_root_of_zero() {
    let x = 0.0;
    assert_eq!(calculator::get_squre_root(x), 0.0);
    assert_that!(calculator::get_squre_root(x), equal_to(0.0));
}

#[test]
pub fn test_square_root_of_one() {
    let x = 1.0;
    assert_eq!(calculator::get_squre_root(x), 1.0);
    assert_that!(calculator::get_squre_root(x), equal_to(1.0));
}
```

For `question4.rs`:

```rust
use super::*;
use hamcrest::prelude::*;
use rand::*;

#[test]
pub fn test_basic_roots() {
    // y = x^2 + 6x - 7
    let a = 1.0;
    let b = 6.0;
```

```
    let c = -7.0;
    assert_that!(calculator::get_roots(a, b, c), equal_to((1.0, -7.0)));
}
#[test]
pub fn test_single_root() {
    // y = x^2
    let a = 1.0;
    let b = 0.0;
    let c = 0.0;
    assert_that!(calculator::get_roots(a, b, c), equal_to((0.0, 0.0)));

}
#[test]
pub fn test_random_solvable_quadratic() {
    let a = rand::thread_rng().gen_range(0.0..2.0);
    let b = rand::thread_rng().gen_range(20.0..100.0);
    let c = rand::thread_rng().gen_range(0.0..10.0);
    let res = calculator::get_roots(a, b, c);
    assert_that!((a * res.0 * res.0 + b * res.0 + c).trunc(), equal_to(0.0));
}
#[test]
#[should_panic]
pub fn test_random_non_solvable_quadratic() {
    let a = rand::thread_rng().gen_range(2.0..10.0);
    let b = rand::thread_rng().gen_range(0.0..5.0);
    let c = rand::thread_rng().gen_range(10.0..20.0);
    assert_that!(calculator::get_roots(a, b, c), equal_to((0.0, 0.0)));
}
```

For the output:

```
coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/UTApp_hamcrest$ cargo test
   Compiling UTApp v0.1.0 (/home/coder/personalProj/rusttest/Assign3/UTApp_hamcrest)

running 20 tests
test question1::add_negative_number ... ok
test question1::add_random_numbers ... ok
test question1::basic_add ... ok
test question1::basic_subtract ... ok
test question1::subtract_negative_number ... ok
test question1::subtract_random_numbers ... ok
test question2::basic_divide ... ok
test question2::basic_multiply ... ok
test question2::divide_negative_number ... ok
test question2::divide_random_numbers ... ok
test question2::multiply_negative_number ... ok
test question2::multiply_random_numbers ... ok
test question3::test_random_negitive_square_root ... ok
test question3::test_random_positive_square_root ... ok
test question3::test_square_root_of_one ... ok
test question3::test_square_root_of_zero ... ok
test question4::test_basic_roots ... ok
test question4::test_random_non_solvable_quadratic - should panic ... ok
test question4::test_random_solvable_quadratic ... ok
test question4::test_single_root ... ok

test result: ok. 20 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in
0.02s
```

Question 4

For `main.rs`:

```rust
use std::io;

fn main() {
    let user_input = get_input();
    println!("the result is {}", function(user_input.0, user_input.1));
}

// define the factorial formula
pub fn function(a: i32, b: i32) -> f64 {
    let formula = (factorial(a as f64)) / (factorial(a as f64 - b as f64) * factorial(b as f64));
    formula
}

// the logic of factorial calculation
pub fn factorial(x: f64) -> f64 {
    match x {
        0.0 => 0.0,
        1.0 => 1.0,
        _ => factorial(x - 1.0) * x,
    }
}

// check whether the input is valid
pub fn check_input(a: &str, b: &str) -> (i32, i32) {
    let input_a: i32 = match a.trim().parse() {
        Ok(num) => num,
        Err(_) => panic!("Can't parse to a number"),
    };

    let input_b: i32 = match b.trim().parse::<i32>() {
        Ok(num) => num,
        Err(_) => panic!("Can't parse to a integer number"),
    };

    if input_a > 0 && input_b > 0 {
        if input_a > input_b {
            (input_a, input_b)
        } else {
            panic!("1st number must bigger than 2nd number!")
        }
    } else {
        panic!("Both number should bigger than zero!")
    }
}

// handle the user input and return the numbers
pub fn get_input() -> (i32, i32) {
    println!("input the first number:");
    let mut input_a = String::new();
    io::stdin().read_line(&mut input_a).expect("Can not read!");

    println!("input the second number");
    let mut input_b = String::new();
    io::stdin().read_line(&mut input_b).expect("Can not read!");

```

```
    let res = check_input(&input_a, &input_b);

    res
}

#[cfg(test)]
mod test;
```

The output is:

```
coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T4$ cargo run
   Compiling A3T4 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T4)

input the first number:
5
input the second number:
3
the result is 10
```

For `test.rs`:

```
#[test]
fn compare() {
    use super::*;
    let res = check_input("8", "6");  // we assume the input is 8 and 6.
    assert!(res.0 > res.1);
}

#[test]
#[should_panic]
fn all_positive_integer() {
    use super::*;

    let res_0 = check_input("6.7", "5.4");  // we assume the input is 6.7 and 5.4.
    // Since the number should be integers, so, it will panic.
    // In other words, it will pass the test
    assert!(res_0.0 > 0 && res_0.1 > 0);

    let res_1 = check_input("-1", "5");  // we assume the input is -1 and 5.
    // Since they both should bigger than zero, so it should panic.
    // In other words, it should pass the test.
    assert!(res_1.0 > 0 && res_1.1 > 0);
}
```

The output is:

```
coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T4$ cargo test
   Compiling A3T4 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T4)

running 2 tests
test test::all_positive_integer - should panic ... ok
test test::compare ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s
```

Question 5

For `main.rs`:

```rust
use std::io;

fn main() {
    let input = handle_input();
    let taxed_income = calculate_tax(input);
    println!("Income after tax is {}", taxed_income);
}

pub fn calculate_tax(income: i32) -> f64 {
    let mut taxed_income: f64 = 0.0;
    let income = income as f64;
    if income >= 0.0 && income < 10000.0 {
        taxed_income = income;
    } else if income >= 10000.0 && income < 50000.0 {
        taxed_income = income - income * 0.1;
    } else if income >= 50000.0 && income < 100000.0 {
        taxed_income = income - income * 0.2;
    } else if income >= 100000.0 && income < 1000000.0 {
        taxed_income = income - income * 0.3;
    } else if income >= 1000000.0 {
        taxed_income = income - income * 0.4
    }
    taxed_income
}

pub fn handle_input() -> i32 {
    println!("Please input your income");
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Can not read");
    let res = is_valid(&input);
    res
}

pub fn is_valid(input: &str) -> i32 {
    let input: i32 = match input.trim().parse() {
        Ok(num) => num,
        Err(_) => panic!("Error parsing input to a integer"),
    };

    if input >= 0 {
        input
    } else {
        panic!("Input is should bigger than or equal to zero!")
    }
}

#[cfg(test)]
mod tests;
```

For the output:

```
coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T5$ cargo run
   Compiling A3T5 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T5)
    Finished dev [unoptimized + debuginfo] target(s) in 0.84s
     Running `target/debug/A3T5`
Please input your income
45320
Income after tax is 40788
```

For `tests.rs`:

```rust
#[test]
#[should_panic]
fn is_negative() {
    use super::*;
    let input = is_valid(&"-24500");
    assert!(input > 0);
}

#[test]
#[should_panic]
fn is_not_integer() {
    use super::*;
    let input = is_valid(&"4500.35");
    assert!(input > 0);
}
```

For the output:

```
coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T5$ cargo test
   Compiling A3T5 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T5)
    Finished test [unoptimized + debuginfo] target(s) in 1.00s
     Running unittests (target/debug/deps/A3T5-86b0f77ab42e08df)

running 2 tests
test tests::is_negative - should panic ... ok
test tests::is_not_integer - should panic ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in
0.01s
```