

# ECE 522 Assignment 3

Zhaoyi Wang 1689747

## *Question 1*

For calculator.rs:

```
1 use std::ops::*;
2
3 pub fn add(x: f64, y: f64) -> f64 {
4     x + y
5 }
6
7 pub fn subtract(x: f64, y: f64) -> f64 {
8     x - y
9 }
10
11 pub fn multiply<T: Mul>(x: T, y: T) -> T::Output {
12     x * y
13 }
14
15 pub fn divide<T: Div>(x: T, y: T) -> T::Output {
16     x / y
17 }
18
19 pub fn get_squire_root(x: f64) -> f64 {
20     if x < 0.0 {
21         panic!("Negative numbers don't have real square roots!");
22     }
23     x.sqrt()
24 }
25
26 pub fn get_roots(a: f64, b: f64, c: f64) -> (f64, f64) {
27     // quadratic equations: ax^2+bx+c=0
28     // (-b + sqrt(b^2-4ac))/(2a) or (-b - sqrt(b^2-4ac))/(2a)
29     let delta = b * b - 4.0 * a * c;
30     if delta >= 0.0 {
31         let root_one = (-b + delta.sqrt()) / (2.0 * a);
32         let root_two = (-b - delta.sqrt()) / (2.0 * a);
33         (root_one, root_two)
34     } else {
35         panic!("There is no root for this equation!");
36         println!("There is no root for this equation!");
37     }
38 }
39
```

For question1.rs:

```

1 use rand::prelude::*;
2
3 // Note this useful idiom: importing names from outer (for mod tests) scope.
4 use super::*;
5
6 #[test]
7 pub fn basic_add() {
8     assert_eq!(calculator::add(1.0, 2.0), 3.0);
9 }
10 #[test]
11 pub fn add_negative_number() {
12     assert_eq!(calculator::add(-1.0, 2.0), 1.0);
13 }
14 #[test]
15 pub fn add_random_numbers() {
16     let mut rng = thread_rng();
17     if rng.gen() { // random bool
18         let x: f64 = rng.gen(); // random number in range [0, 1)
19         let y: f64 = rng.gen();
20         assert_eq!(calculator::add(x, y), x+y);
21     }
22 }
23
24 #[test]
25 pub fn basic_subtract() {
26     assert_eq!(calculator::subtract(4.0, 2.0), 2.0);
27 }
28 #[test]
29 pub fn subtract_negative_number() {
30     assert_eq!(calculator::subtract(-3.0, 2.0), -5.0);
31 }
32 #[test]
33 pub fn subtract_random_numbers() {
34     let mut rng = thread_rng();
35     if rng.gen() { // random bool
36         let x: f64 = rng.gen(); // random number in range [0, 1)
37         let y: f64 = rng.gen();
38         assert_eq!(calculator::subtract(x, y), x-y);
39     }
40 }

```

For question2.rs:

```

1 use super::*;
2 use rand::prelude::*;
3
4 #[test]
5 pub fn basic_multiply() {
6     let x = 4;
7     let y = 2;
8     assert_eq!(calculator::multiply(x, y), 8);
9 }
10
11 #[test]
12 pub fn multiply_negative_number() {
13     let x = -4;
14     let y = -2;
15     assert_eq!(calculator::multiply(x, y), 8)
16 }
17

```

```

18 #[test]
19 pub fn multiply_random_numbers() {
20     let mut rng = thread_rng();
21     if rng.gen() {
22         // random bool
23         let x: f64 = rng.gen(); // random number in range [0, 1)
24         let y: f64 = rng.gen();
25         assert_eq!(calculator::multiply(x, y), x * y);
26     }
27 }
28
29 #[test]
30 pub fn basic_divide() {
31     let x = 2;
32     let y = 2;
33     assert_eq!(calculator::divide(x, y), 1);
34 }
35
36 #[test]
37 pub fn divide_negative_number() {
38     let x = -2;
39     let y = -2;
40     assert_eq!(calculator::divide(x, y), 1);
41 }
42 #[test]
43 pub fn divide_random_numbers() {
44     let mut rng = thread_rng();
45     if rng.gen() {
46         // random bool
47         let x: f64 = rng.gen(); // random number in range [0, 1)
48         let y: f64 = rng.gen();
49         assert_eq!(calculator::divide(x, y), x / y);
50     }
51 }

```

For question3.rs:

```

1 use super::*;
2 use rand::*;
3
4 #[test]
5 pub fn test_random_positive_square_root() {
6     // let rng = rand::thread_rng().gen::<f64>();
7     let rng = rand::thread_rng().gen_range(0.0..100.0);
8     assert_eq!(calculator::get_squire_root(rng), rng.sqrt());
9 }
10 #[test]
11 #[should_panic]
12 pub fn test_random_negative_square_root() {
13     let rng = rand::thread_rng().gen_range(-100.0..0.0);
14     assert_eq!(calculator::get_squire_root(rng), (rng).sqrt());
15 }
16 #[test]
17 pub fn test_square_root_of_zero() {
18     let x = 0.0;
19     assert_eq!(calculator::get_squire_root(x), 0.0);
20 }
21
22 #[test]
23 pub fn test_square_root_of_one() {

```

```

24     let x = 1.0;
25     assert_eq!(calculator::get_squire_root(x), 1.0);
26 }

```

For question4.rs:

```

1  use super::*;
2  use rand::*;
3
4  #[test]
5  pub fn test_basic_roots() {
6      // y = x^2 + 6x - 7
7      let a = 1.0;
8      let b = 6.0;
9      let c = -7.0;
10     assert_eq!(calculator::get_roots(a, b, c), (1.0, -7.0));
11 }
12 #[test]
13 pub fn test_single_root() {
14     // y = x^2
15     let a = 1.0;
16     let b = 0.0;
17     let c = 0.0;
18     assert_eq!(calculator::get_roots(a, b, c), (0.0, 0.0));
19 }
20 #[test]
21 pub fn test_random_solvable_quadratic() {
22     let a = rand::thread_rng().gen_range(0.0..2.0);
23     let b = rand::thread_rng().gen_range(20.0..100.0);
24     let c = rand::thread_rng().gen_range(0.0..10.0);
25     let res = calculator::get_roots(a,b,c);
26     assert_eq!((a*res.0*res.0+b*res.0+c).trunc(),0.0);
27 }
28 #[test]
29 #[should_panic]
30 pub fn test_random_non_solvable_quadratic() {
31     let a = rand::thread_rng().gen_range(2.0..10.0);
32     let b = rand::thread_rng().gen_range(0.0..5.0);
33     let c = rand::thread_rng().gen_range(10.0..20.0);
34     assert_eq!(calculator::get_roots(a, b, c), (0.0, 0.0));
35 }

```

For the output:

```

1  coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3$ cargo test
2      Compiling UApp v0.1.0 (/home/coder/personalProj/rusttest/Assign3)
3
4  running 20 tests
5  test question1::add_negative_number ... ok
6  test question1::add_random_numbers ... ok
7  test question1::basic_add ... ok
8  test question1::basic_subtract ... ok
9  test question1::subtract_negative_number ... ok
10 test question1::subtract_random_numbers ... ok
11 test question2::basic_divide ... ok
12 test question2::basic_multiply ... ok
13 test question2::divide_negative_number ... ok
14 test question2::divide_random_numbers ... ok
15 test question2::multiply_negative_number ... ok

```

```

16 test question2::multiply_random_numbers ... ok
17 test question3::test_random_negative_square_root ... ok
18 test question3::test_random_positive_square_root ... ok
19 test question3::test_square_root_of_one ... ok
20 test question3::test_square_root_of_zero ... ok
21 test question4::test_basic_roots ... ok
22 test question4::test_random_non_solvable_quadratic - should panic ... ok
23 test question4::test_random_solvable_quadratic ... ok
24 test question4::test_single_root ... ok
25
26 test result: ok. 20 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in
    0.02s

```

## Question 2

For main.rs:

```

1 mod question;
2 mod player;
3
4 #[cfg(test)] #[macro_use] extern crate hamcrest;
5
6 fn main() {}

```

For player.rs:

```

1 pub struct Player {
2     pub id: i32,
3     pub first_name: String,
4     pub last_name: String,
5 }

```

For question.rs:

```

1 use super::*;
2 use hamcrest::*;
3
4 #[test]
5 fn name_is_String() {
6     let player_one = player::Player {
7         id: 001,
8         first_name: String::from("Tim"),
9         last_name: String::from("White"),
10    };
11
12    let player_two = player::Player {
13        id: 002,
14        first_name: String::from("Mike"),
15        last_name: String::from("Thompson"),
16    };
17
18    assert_that!(player_one.first_name, is(type_of::<String>()));
19    assert_that!(player_one.last_name, is(type_of::<String>()));
20 }
21 #[test]

```

```

22 fn all_the_same(){
23     let player_one = player::Player {
24         id: 001,
25         first_name: String::from("Tim"),
26         last_name: String::from("White"),
27     };
28
29     let player_two = player::Player {
30         id: 001,
31         first_name: String::from("Tim"),
32         last_name: String::from("White"),
33     };
34
35     assert_that!(player_one.id, is(equal_to(player_two.id)));
36     assert_that!(player_one.first_name, is(equal_to(player_two.first_name)));
37     assert_that!(player_one.last_name, is(equal_to(player_two.last_name)));
38 }

```

For the output:

```

1  coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T2$ cargo test
2      Compiling A3T2 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T2)
3
4  running 2 tests
5  test question::all_the_same ... ok
6  test question::name_is_String ... ok
7
8  test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s

```

## Question 3

For question1.rs:

```

1  use hamcrest::prelude::*;
2  use rand::prelude::*;
3
4  // Note this useful idiom: importing names from outer (for mod tests) scope.
5  use super::*;
6
7  #[test]
8  pub fn basic_add() {
9      assert_that!(calculator::add(1.0, 2.0), is(equal_to(3.0)));
10 }
11 #[test]
12 pub fn add_negative_number() {
13     assert_that!(calculator::add(-1.0, 2.0), is(equal_to(1.0)));
14 }
15 #[test]
16 pub fn add_random_numbers() {
17     let mut rng = thread_rng();
18     if rng.gen() {
19         // random bool
20         let x: f64 = rng.gen(); // random number in range [0, 1)
21         let y: f64 = rng.gen();
22         assert_that!(calculator::add(x, y), is(equal_to(x + y)));
23     }

```

```

24 }
25
26 #[test]
27 pub fn basic_subtract() {
28     assert_that!(calculator::subtract(4.0, 2.0), is(equal_to(2.0)));
29 }
30 #[test]
31 pub fn subtract_negative_number() {
32     assert_that!(calculator::subtract(-3.0, 2.0), is(equal_to(-5.0)));
33 }
34 #[test]
35 pub fn subtract_random_numbers() {
36     let mut rng = thread_rng();
37     if rng.gen() {
38         // random bool
39         let x: f64 = rng.gen(); // random number in range [0, 1)
40         let y: f64 = rng.gen();
41         assert_that!(calculator::subtract(x, y), is(equal_to(x - y)));
42     }
43 }

```

For question2.rs:

```

1 use super::*;
2 use hamcrest::prelude::*;
3 use rand::prelude::*;
4
5 #[test]
6 pub fn basic_multiply() {
7     let x = 4;
8     let y = 2;
9     assert_that!(calculator::multiply(x, y), is(equal_to(8)));
10 }
11
12 #[test]
13 pub fn multiply_negative_number() {
14     let x = -4;
15     let y = -2;
16     assert_that!(calculator::multiply(x, y), is(equal_to(8)));
17 }
18
19 #[test]
20 pub fn multiply_random_numbers() {
21     let mut rng = thread_rng();
22     if rng.gen() {
23         // random bool
24         let x: f64 = rng.gen(); // random number in range [0, 1)
25         let y: f64 = rng.gen();
26         assert_that!(calculator::multiply(x, y), is(equal_to(x * y)));
27     }
28 }
29
30 #[test]
31 pub fn basic_divide() {
32     let x = 2;
33     let y = 2;
34     assert_that!(calculator::divide(x, y), is(equal_to(1)));
35 }
36
37 #[test]

```

```

38 pub fn divide_negative_number() {
39     let x = -2;
40     let y = -2;
41     assert_that!(calculator::divide(x, y), is(equal_to(1)));
42 }
43 #[test]
44 pub fn divide_random_numbers() {
45     let mut rng = thread_rng();
46     if rng.gen() {
47         // random bool
48         let x: f64 = rng.gen(); // random number in range [0, 1)
49         let y: f64 = rng.gen();
50         assert_that!(calculator::divide(x, y), is(equal_to(x / y)));
51     }
52 }

```

For question3.rs:

```

1 use super::*;
2 use rand::*;
3 use hamcrest::prelude::*;
4
5 #[test]
6 pub fn test_random_positive_square_root() {
7     // let rng = rand::thread_rng().gen::<f64>();
8     let rng = rand::thread_rng().gen_range(0.0..100.0);
9     assert_that!(calculator::get_squire_root(rng), equal_to((rng as f64).sqrt()));
10 }
11 #[test]
12 #[should_panic]
13 pub fn test_random_negative_square_root() {
14     let rng = rand::thread_rng().gen_range(-100.0..0.0);
15     assert_that!(calculator::get_squire_root(rng), equal_to((rng as f64).sqrt()));
16 }
17 #[test]
18 pub fn test_square_root_of_zero() {
19     let x = 0.0;
20     assert_eq!(calculator::get_squire_root(x), 0.0);
21     assert_that!(calculator::get_squire_root(x), equal_to(0.0));
22 }
23
24 #[test]
25 pub fn test_square_root_of_one() {
26     let x = 1.0;
27     assert_eq!(calculator::get_squire_root(x), 1.0);
28     assert_that!(calculator::get_squire_root(x), equal_to(1.0));
29 }

```

For question4.rs:

```

1 use super::*;
2 use hamcrest::prelude::*;
3 use rand::*;
4
5 #[test]
6 pub fn test_basic_roots() {
7     // y = x^2 + 6x - 7
8     let a = 1.0;
9     let b = 6.0;

```



```

10     let c = -7.0;
11     assert_that!(calculator::get_roots(a, b, c), equal_to((1.0, -7.0)));
12 }
13 #[test]
14 pub fn test_single_root() {
15     // y = x^2
16     let a = 1.0;
17     let b = 0.0;
18     let c = 0.0;
19     assert_that!(calculator::get_roots(a, b, c), equal_to((0.0, 0.0)));
20 }
21 }
22 #[test]
23 pub fn test_random_solvable_quadratic() {
24     let a = rand::thread_rng().gen_range(0.0..2.0);
25     let b = rand::thread_rng().gen_range(20.0..100.0);
26     let c = rand::thread_rng().gen_range(0.0..10.0);
27     let res = calculator::get_roots(a, b, c);
28     assert_that!((a * res.0 * res.0 + b * res.0 + c).trunc(), equal_to(0.0));
29 }
30 #[test]
31 #[should_panic]
32 pub fn test_random_non_solvable_quadratic() {
33     let a = rand::thread_rng().gen_range(2.0..10.0);
34     let b = rand::thread_rng().gen_range(0.0..5.0);
35     let c = rand::thread_rng().gen_range(10.0..20.0);
36     assert_that!(calculator::get_roots(a, b, c), equal_to((0.0, 0.0)));
37 }

```

For the output:

```

1  coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/UTApp_hamcrest$ cargo test
2  Compiling UTApp v0.1.0 (/home/coder/personalProj/rusttest/Assign3/UTApp_hamcrest)
3
4  running 20 tests
5  test question1::add_negative_number ... ok
6  test question1::add_random_numbers ... ok
7  test question1::basic_add ... ok
8  test question1::basic_subtract ... ok
9  test question1::subtract_negative_number ... ok
10 test question1::subtract_random_numbers ... ok
11 test question2::basic_divide ... ok
12 test question2::basic_multiply ... ok
13 test question2::divide_negative_number ... ok
14 test question2::divide_random_numbers ... ok
15 test question2::multiply_negative_number ... ok
16 test question2::multiply_random_numbers ... ok
17 test question3::test_random_negative_square_root ... ok
18 test question3::test_random_positive_square_root ... ok
19 test question3::test_square_root_of_one ... ok
20 test question3::test_square_root_of_zero ... ok
21 test question4::test_basic_roots ... ok
22 test question4::test_random_non_solvable_quadratic - should panic ... ok
23 test question4::test_random_solvable_quadratic ... ok
24 test question4::test_single_root ... ok
25
26 test result: ok. 20 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in
    0.02s

```

## Question 4

For main.rs:

```
1 use std::io;
2
3 fn main() {
4     let user_input = get_input();
5     println!("the result is {}", function(user_input.0, user_input.1));
6 }
7
8 // define the factorial formula
9 pub fn function(a: i32, b: i32) -> f64 {
10     let formula = (factorial(a as f64)) / (factorial(a as f64 - b as f64) * factorial(b as
11     formula
12 }
13
14 // the logic of factorial calculation
15 pub fn factorial(x: f64) -> f64 {
16     match x {
17         0.0 => 0.0,
18         1.0 => 1.0,
19         _ => factorial(x - 1.0) * x,
20     }
21 }
22
23 // check whether the input is valid
24 pub fn check_input(a: &str, b: &str) -> (i32, i32) {
25     let input_a: i32 = match a.trim().parse() {
26         Ok(num) => num,
27         Err(_) => panic!("Can't parse to a number"),
28     };
29
30     let input_b: i32 = match b.trim().parse::<i32>() {
31         Ok(num) => num,
32         Err(_) => panic!("Can't parse to a integer number"),
33     };
34
35     if input_a > 0 && input_b > 0 {
36         if input_a > input_b {
37             (input_a, input_b)
38         } else {
39             panic!("1st number must bigger than 2nd number!")
40         }
41     } else {
42         panic!("Both number should bigger than zero!")
43     }
44 }
45
46 // handle the user input and return the numbers
47 pub fn get_input() -> (i32, i32) {
48     println!("input the first number:");
49     let mut input_a = String::new();
50     io::stdin().read_line(&mut input_a).expect("Can not read!");
51
52     println!("input the second number");
53     let mut input_b = String::new();
54     io::stdin().read_line(&mut input_b).expect("Can not read!");
55 }
```

```

56     let res = check_input(&input_a, &input_b);
57
58     res
59 }
60
61 #[cfg(test)]
62 mod test;

```

The output is:

```

1  coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T4$ cargo run
2      Compiling A3T4 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T4)
3
4  input the first number:
5  5
6  input the second number:
7  3
8  the result is 10

```

For test.rs:

```

1  #[test]
2  fn compare() {
3      use super::*;
4      let res = check_input("8", "6"); // we assume the input is 8 and 6.
5      assert!(res.0 > res.1);
6  }
7
8  #[test]
9  #[should_panic]
10 fn all_positive_integer() {
11     use super::*;
12
13     let res_0 = check_input("6.7", "5.4"); // we assume the input is 6.7 and 5.4.
14     // Since the number should be integers, so, it will panic.
15     // In other words, it will pass the test
16     assert!(res_0.0 > 0 && res_0.1 > 0);
17
18     let res_1 = check_input("-1", "5"); // we assume the input is -1 and 5.
19     // Since they both should bigger than zero, so it should panic.
20     // In other words, it should pass the test.
21     assert!(res_1.0 > 0 && res_1.1 > 0);
22 }

```

The output is:

```

1  coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T4$ cargo test
2      Compiling A3T4 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T4)
3
4  running 2 tests
5  test test::all_positive_integer - should panic ... ok
6  test test::compare ... ok
7
8  test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s

```

## Question 5

For main.rs:

```
1 use std::io;
2
3 fn main() {
4     let input = handle_input();
5     let taxed_income = calculate_tax(input);
6     println!("Income after tax is {}", taxed_income);
7 }
8
9 pub fn calculate_tax(income: i32) -> f64 {
10     let mut taxed_income: f64 = 0.0;
11     let income = income as f64;
12     if income >= 0.0 && income < 10000.0 {
13         taxed_income = income;
14     } else if income >= 10000.0 && income < 50000.0 {
15         taxed_income = income - income * 0.1;
16     } else if income >= 50000.0 && income < 100000.0 {
17         taxed_income = income - income * 0.2;
18     } else if income >= 100000.0 && income < 1000000.0 {
19         taxed_income = income - income * 0.3;
20     } else if income >= 1000000.0 {
21         taxed_income = income - income * 0.4
22     }
23     taxed_income
24 }
25
26 pub fn handle_input() -> i32 {
27     println!("Please input your income");
28     let mut input = String::new();
29     io::stdin().read_line(&mut input).expect("Can not read");
30     let res = is_valid(&input);
31     res
32 }
33
34 pub fn is_valid(input: &str) -> i32 {
35     let input: i32 = match input.trim().parse() {
36         Ok(num) => num,
37         Err(_) => panic!("Error parsing input to a integer"),
38     };
39
40     if input >= 0 {
41         input
42     } else {
43         panic!("Input is should bigger than or equal to zero!")
44     }
45 }
46
47 #[cfg(test)]
48 mod tests;
```

For the output:

```
1 coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T5$ cargo run
2   Compiling A3T5 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T5)
3   Finished dev [unoptimized + debuginfo] target(s) in 0.84s
4   Running `target/debug/A3T5`
5 Please input your income
6 45320
7 Income after tax is 40788
```

For tests.rs:

```
1  #[test]
2  #[should_panic]
3  fn is_negative() {
4      use super::*;
5      let input = is_valid(&"-24500");
6      assert!(input > 0);
7  }
8
9  #[test]
10 #[should_panic]
11 fn is_not_integer() {
12     use super::*;
13     let input = is_valid(&"4500.35");
14     assert!(input > 0);
15 }
```

For the output:

```
1 coder@ubuntu-s-1vcpu-2gb-tor1-01:~/personalProj/rusttest/Assign3/A3T5$ cargo test
2   Compiling A3T5 v0.1.0 (/home/coder/personalProj/rusttest/Assign3/A3T5)
3   Finished test [unoptimized + debuginfo] target(s) in 1.00s
4   Running unittests (target/debug/deps/A3T5-86b0f77ab42e08df)
5
6 running 2 tests
7 test tests::is_negative - should panic ... ok
8 test tests::is_not_integer - should panic ... ok
9
10 test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in
    0.01s
```