ECE 326 Tutorial 8

Rust & Exercise 7 Review

Prepared by: Wendy Qiu
October 21, 2020

Exercise 7 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

1. Rust enum is an example of algebraic type.

T **F**

Algebraic data type: composite type

- Product types: tuples
- Sum types: unions & enum

Exercise 7 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

2. The purpose of generic programming is code reuse across different data types.

T **F**

- Minimal assumptions about the structure of data
- Maximize code reuse

Exercise 7 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

3. Box forces the contained object to be heap allocated

T **F**

An allocator that provides malloc-like functionality

Exercise 7 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

4. Lifetime parameter must be added to all structures with non-static references.

T **F**

- **Generic lifetime parameter**: lifetime information to function signature
- **Static lifetime reference**: entire duration of the program

Exercise 7 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

5. Lifetime elision optimizes the binary by eliminating the need to copy parameters.

T **F**

Exercise 7 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following statements are true regarding ownership and borrowing?

i. You cannot access a variable after it has been moved. ➔ **True**

Exercise 7 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following statements are true regarding ownership and borrowing?

ii. You cannot change a variable while there are immutable references to it.

➔ **True**

Exercise 7 – Q2a: (Code Example)

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following statements are true regarding ownership and borrowing?

iii. You can mutably borrow multiple non-overlapping slices of a sequence at one time.

➔ **True**

Exercise 7 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following statements are true regarding ownership and borrowing?

iv. You can have multiple immutable references to a variable at one time

➔ **True**

Exercise 7 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following statements are true regarding ownership and borrowing?

v. You can change an immutable variable into a mutable one after moving it

➔ **True**

Exercise 7 – Q2b:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

b) Which of the following traits have default implementation?

i. PartialEq ii. Display iii. Not iv. Clone v. Default

Exercise 7 – Q3-a:

a) Similarities and differences: Python mixins & Rust traits

Similarities:

- Both can supply default implementation.
- A structure/class can have multiple mixins or multiple traits.
- Both can be used for code reuse.

Differences:

- Traits cannot have data fields, mixins can.
- Traits are not affected by order of inheritance, mixins can be.
- Traits are supposed to be implemented, mixins are supposed to be included.
- Traits provides an interface, mixins only provides implementation.

Exercise 7 – Q3-b:

b) Given the following codes, if you try to compile it directly, it will generate errors/warnings.

Identify the line(s) where error(s) would occur by specifying the line number(s), and if possible, write the code that would fix it.

If a print statement would cause the error, write “DELETE” in the table.

```
1    fn create_model(company: String) -> String {
2        let model = String::from("Default, ");
3        if company == "Boeing" {
4            model = String::from("747, ");
5        }
6        else if company == "Airbus" {
7            model = String::from("A330, ");
8        }
9        model.push_str(&company);
10       model
11   }
12
13   fn create_airbus() -> &str {
14       let some_string = "Airbus";
15       &some_string
16   }
17
18   fn create_plane() {
19       let plane1 = create_airbus();
20       let plane2 = String::from("Boeing");
21       let plane3 = create_model(plane1);
22       let plane4 = create_model(plane2);
23       println!("Plane 1: {}", plane1);
24       println!("Plane 2: {}", plane2);
25       println!("Plane 3: {}", plane3);
26       println!("Plane 4: {}", plane4);
27   }
```

let **mut** model = String::from("Default, ");

```
29   fn main() {
30       let mut height = 100;
31       let my_plane = "777, Boeing";
32       {
33           let b = my_plane;
34       }
35       println!("Plane is a: {}", b);
36       println!("Currently at {}", height);
37       {
38           let c = 2;
39           height += c;
40       }
41       println!("Ascending to {}", height);
42       create_plane();
43   }
```

```
1   fn create_model(company: String) -> String {
2       let model = String::from("Default, ");
3       if company == "Boeing" {
4           model = String::from("747, ");
5       }
6       else if company == "Airbus" {
7           model = String::from("A330, ");
8       }
9       model.push_str(&company);
10      model
11  }
12
13  fn create_airbus() -> &str {
14      let some_string = "Airbus";
15      &some_string
16  }
17
18  fn create_plane() {
19      let plane1 = create_airbus();
20      let plane2 = String::from("Boeing");
21      let plane3 = create_model(plane1);
22      let plane4 = create_model(plane2);
23      println!("Plane 1: {}", plane1);
24      println!("Plane 2: {}", plane2);
25      println!("Plane 3: {}", plane3);
26      println!("Plane 4: {}", plane4);
27  }
```

fn create_airbus () -> &**'static** str {

```
29  fn main() {
30      let mut height = 100;
31      let my_plane = "777, Boeing";
32      {
33          let b = my_plane;
34      }
35      println!("Plane is a: {}", b);
36      println!("Currently at {}", height);
37      {
38          let c = 2;
39          height += c;
40      }
41      println!("Ascending to {}", height);
42      create_plane();
43  }
```

```
1    fn create_model(company: String) -> String {
2        let model = String::from("Default, ");
3        if company == "Boeing" {
4            model = String::from("747, ");
5        }
6        else if company == "Airbus" {
7            model = String::from("A330, ");
8        }
9        model.push_str(&company);
10       model
11   }
12
13   fn create_airbus() -> &str {
14       let some_string = "Airbus";
15       &some_string
16   }
17
18   fn create_plane() {
19       let plane1 = create_airbus();
20       let plane2 = String::from("Boeing");
21       let plane3 = create_model(plane1);
22       let plane4 = create_model(plane2);
23       println!("Plane 1: {}", plane1);
24       println!("Plane 2: {}", plane2);
25       println!("Plane 3: {}", plane3);
26       println!("Plane 4: {}", plane4);
27   }
```

let plane3 = create_model(plane1**.to_string()**);

```
29   fn main() {
30       let mut height = 100;
31       let my_plane = "777, Boeing";
32       {
33           let b = my_plane;
34       }
35       println!("Plane is a: {}", b);
36       println!("Currently at {}", height);
37       {
38           let c = 2;
39           height += c;
40       }
41       println!("Ascending to {}", height);
42       create_plane();
43   }
```

```
1   fn create_model(company: String) -> String {
2       let model = String::from("Default, ");
3       if company == "Boeing" {
4           model = String::from("747, ");
5       }
6       else if company == "Airbus" {
7           model = String::from("A330, ");
8       }
9       model.push_str(&company);
10      model
11  }
12
13  fn create_airbus() -> &str {
14      let some_string = "Airbus";
15      &some_string
16  }
17
18  fn create_plane() {
19      let plane1 = create_airbus();
20      let plane2 = String::from("Boeing");
21      let plane3 = create_model(plane1);
22      let plane4 = create_model(plane2);
23      println!("Plane 1: {}", plane1);
24      println!("Plane 2: {}", plane2);
25      println!("Plane 3: {}", plane3);
26      println!("Plane 4: {}", plane4);
27  }
```

DELETE

```
29  fn main() {
30      let mut height = 100;
31      let my_plane = "777, Boeing";
32      {
33          let b = my_plane;
34      }
35      println!("Plane is a: {}", b);
36      println!("Currently at {}", height);
37      {
38          let c = 2;
39          height += c;
40      }
41      println!("Ascending to {}", height);
42      create_plane();
43  }
```

```
1    fn create_model(company: String) -> String {
2        let model = String::from("Default, ");
3        if company == "Boeing" {
4            model = String::from("747, ");
5        }
6        else if company == "Airbus" {
7            model = String::from("A330, ");
8        }
9        model.push_str(&company);
10       model
11   }
12
13   fn create_airbus() -> &str {
14       let some_string = "Airbus";
15       &some_string
16   }
17
18   fn create_plane() {
19       let plane1 = create_airbus();
20       let plane2 = String::from("Boeing");
21       let plane3 = create_model(plane1);
22       let plane4 = create_model(plane2);
23       println!("Plane 1: {}", plane1);
24       println!("Plane 2: {}", plane2);
25       println!("Plane 3: {}", plane3);
26       println!("Plane 4: {}", plane4);
27   }
```

(optional) leads to a warning of unused variable ‘b’

```
29   fn main() {
30       let mut height = 100;
31       let my_plane = "777, Boeing";
32       {
33           let b = my_plane;
34       }
35       println!("Plane is a: {}", b);
36       println!("Currently at {}", height);
37       {
38           let c = 2;
39           height += c;
40       }
41       println!("Ascending to {}", height);
42       create_plane();
43   }
```

DELETE

```
1   fn create_model(company: String) -> String {
2       let model = String::from("Default, ");
3       if company == "Boeing" {
4           model = String::from("747, ");
5       }
6       else if company == "Airbus" {
7           model = String::from("A330, ");
8       }
9       model.push_str(&company);
10      model
11  }
12
13  fn create_airbus() -> &str {
14      let some_string = "Airbus";
15      &some_string
16  }
17
18  fn create_plane() {
19      let plane1 = create_airbus();
20      let plane2 = String::from("Boeing");
21      let plane3 = create_model(plane1);
22      let plane4 = create_model(plane2);
23      println!("Plane 1: {}", plane1);
24      println!("Plane 2: {}", plane2);
25      println!("Plane 3: {}", plane3);
26      println!("Plane 4: {}", plane4);
27  }
```

```
29  fn main() {
30      let mut height = 100;
31      let my_plane = "777, Boeing";
32      {
33          let b = my_plane;
34      }
35      println!("Plane is a: {}", b);
36      println!("Currently at {}", height);
37      {
38          let c = 2;
39          height += c;
40      }
41      println!("Ascending to {}", height);
42      create_plane();
43  }
```

DELETE

Exercise 7 – Q3-c:

c) Assuming all the errors were fixed in the above program, write the output of the program:

Currently at 100
Ascending to 102
Plane 1: Airbus
Plane 3: A330, Airbus
Plane 4: 747, Boeing

```
1   fn create_model(company: String) -> String {
2       let model = String::from("Default, ");
3       if company == "Boeing" {
4           model = String::from("747, ");
5       }
6       else if company == "Airbus" {
7           model = String::from("A330, ");
8       }
9       model.push_str(&company);
10      model
11  }
12
13  fn create_airbus() -> &str {
14      let some_string = "Airbus";
15      &some_string
16  }
17
18  fn create_plane() {
19      let plane1 = create_airbus();
20      let plane2 = String::from("Boeing");
21      let plane3 = create_model(plane1);
22      let plane4 = create_model(plane2);
23      println!("Plane 1: {}", plane1);
24      println!("Plane 2: {}", plane2);
25      println!("Plane 3: {}", plane3);
26      println!("Plane 4: {}", plane4);
27  }
```

```
29  fn main() {
30      let mut height = 100;
31      let my_plane = "777, Boeing";
32      {
33          let b = my_plane;
34      }
35      println!("Plane is a: {}", b);
36      println!("Currently at {}", height);
37      {
38          let c = 2;
39          height += c;
40      }
41      println!("Ascending to {}", height);
42      create_plane();
43  }
```

Currently at 100

Ascending to 102

Plane 1: Airbus

Plane 3: A330, Airbus

Plane 4: 747, Boeing

Exercise 7 – Q4 - a: (Code Example)

a) Write structure named **Matrix** which supports a **2x2 matrix** of **type f64**, and implements the **determinant method**, the **transpose method**, and the **inverse method**. The transpose method should modify the existing matrix, but the inverse method should return a new matrix if the matrix is invertible, otherwise it should return None. (Hint: use Option<T>).

Write a new **static method which creates and initializes the matrix**. Implement the **Display trait** for Matrix.

Exercise 7 – Q4 - b: (Code Example)

b) Write a generic function, sum_of_squares, calculates the sum of squares of a generic array slice. *You may need some trait bounds.*

```
fn sum_of_squares<T: Default + AddAssign + Mul<Output=T> + Copy>
    (list: &[T]) -> T
{
    let mut sum = Default::default();
    for &n in list {
        sum += n * n;
    }
    sum
}
```

Exercise 7 – Q4 - c: (Code Example)

c) *Write a function that takes* ***two string slices***, ***text and word***, *and* ***return a vector of all occurrences of the word*** *in text in string slices.*
Hint: look up the find method for a string slice. You may need to “fix” the function signature. You may assume the text to consist of only ascii characters.

```
fn findall(text: &str, word: &str) -> Vec<&str>
```

Exercise 7 – Q4 - d: (Code Example)

d) *Implement a* **sorted singly linked list** *of* ***i64 elements*** *where the insert method will automatically place the new element such that the list remains in* ***ascending order****.*

Complete the **remove method** *such that all elements with the specified value is removed from the list. Implement the* **Display trait** *to print all elements of the list.*

Thanks for listening!