# ECE 326 Tutorial 7

Rust & Exercise 6 Review

Prepared by: Wendy Qiu
October 21, 2020

# Rust Programming

## Small Example

# Exercise 5 - Q1:

**Question 1. True or False**

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

1. One major disadvantage of Rust is that it automatically adds runtime type safety checks, which negatively affects its runtime performance.

**T** **F**

Longer compile time for Rust

# Exercise 5 - Q1:

**Question 1. True or False**

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

2. In Rust, ownership is a mechanism to check for memory leaks at runtime.

**T** **F**

Compiler checks at compile time

# Exercise 5 - Q1:

**Question 1. True or False**

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

3. Rust literals are first class citizens.

**T** **F**

- Can be used: let x = 1u8
- Can be constructed: create literals in local scope
- Have a type: e.g. 8-bit unsigned int

# Exercise 5 - Q1:

**Question 1. True or False**

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

4. The as operator in Python and Rust does the same thing.

**T** **F**

- Rust: as casts one type to another
- Python: e.g. with open … as, from … import … as

# Exercise 5 - Q1:

**Question 1. True or False**

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

5. The Rust compiler will attempt to copy an object only after it decides that moving the object is not permissible..

**T** **F**

Copy before move!

# Exercise 5 – Q2a:

**Question 2. Multiple Choices**

Pick all answer(s) that are correct.

a) Which of the following keyword can be used as part of an expression in Rust?

Options: *if* *let* *loop* *match* *use*

eg. if row.len() == 0
loop { count += 1; }
let binary = match boolean;

# Exercise 5 – Q2b:

**Question 2. Multiple Choices**

Pick all answer(s) that are correct.

b) Which of the following types of bugs are absent from Rust programs?

Rust is memory safe!

➔ No *null pointer exception, dangling pointers* or *buffer overflow*

# Exercise 5 – Q2b:

**Question 2. Multiple Choices**

Pick all answer(s) that are correct.

b) Which of the following types of bugs are absent from Rust programs?

Rust is thread safe!

➔ Prevent *data race*

# Exercise 5 – Q2b:

**Question 2. Multiple Choices**

Pick all answer(s) that are correct.

b) Which of the following types of bugs are absent from Rust programs?

Divide by zero error?

➔ Yes: undefined results

# Exercise 5 – Q3-a:

***a) What is the difference between the following two sets of statements?***

```
// set 1
let x = 5;
let x = x + 2;
let x = x * 3;
```

```
// set 2
let mut x = 5;
x = x + 2;
x = x * 3;
```

# Exercise 5 – Q3-a: (Code Example)

***a) What is the difference between the following two sets of statements?***

```
// set 1
let x = 5;
let x = x + 2;
let x = x * 3;

// set 2
let mut x = 5;
x = x + 2;
x = x * 3;
```

- In set 1, there are three immutable variables with the same name. The second replaces the first and the third replaces the second.
- In set 2, x is a mutable variable, and you are free to change its value.

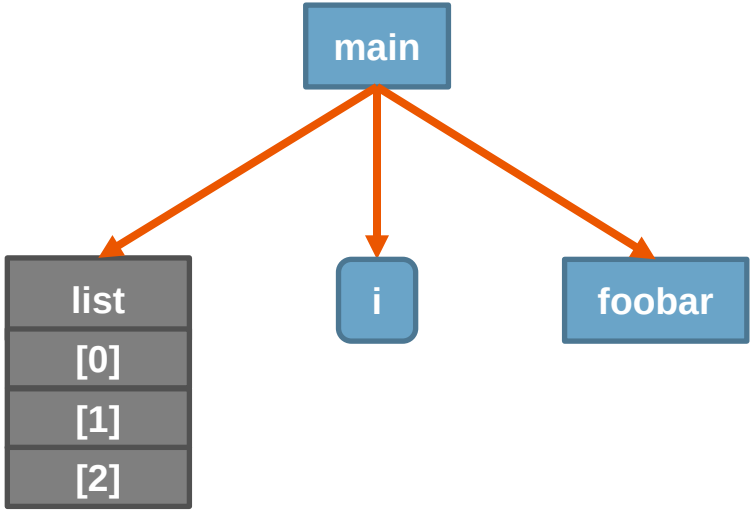
## b) Ownership Diagram:

```
fn foobar(n: i32, list: &Vec<Option<& str>>) {
    for x in {0..n} {
        let mut c = 0;
        for elem in list {
            let val = elem.unwrap();
            println!("{}.{}: {}", x, c, val); /* here */
            c += 1;
        }
    }
}

fn main() {
    let i = 5;
    let list = vec![ Some("hello"), Some("new"), Some("world") ];
    foobar(i, &list);
}
```

## *b) Ownership Diagram:*

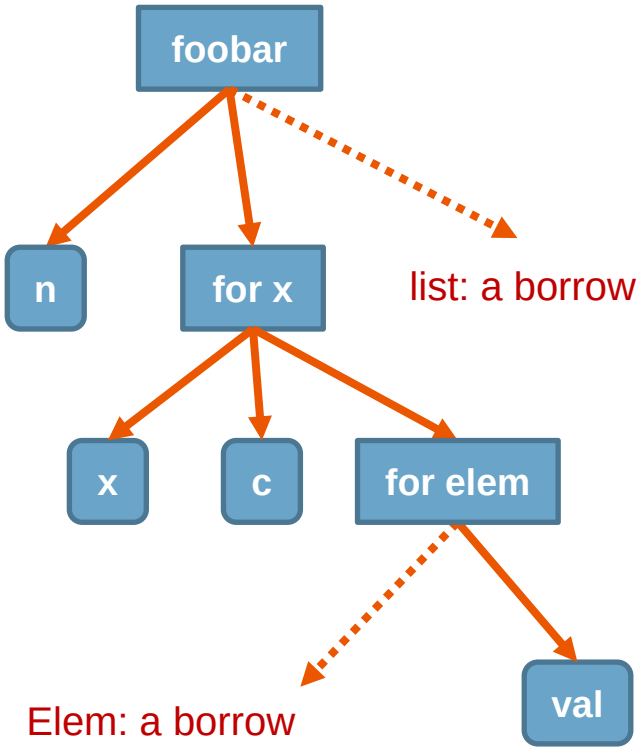
```
fn main() {
    let i = 5;
    let list = vec![Some("hello"), Some("new"), Some("world") ];
    foobar(i, &list);
}
```

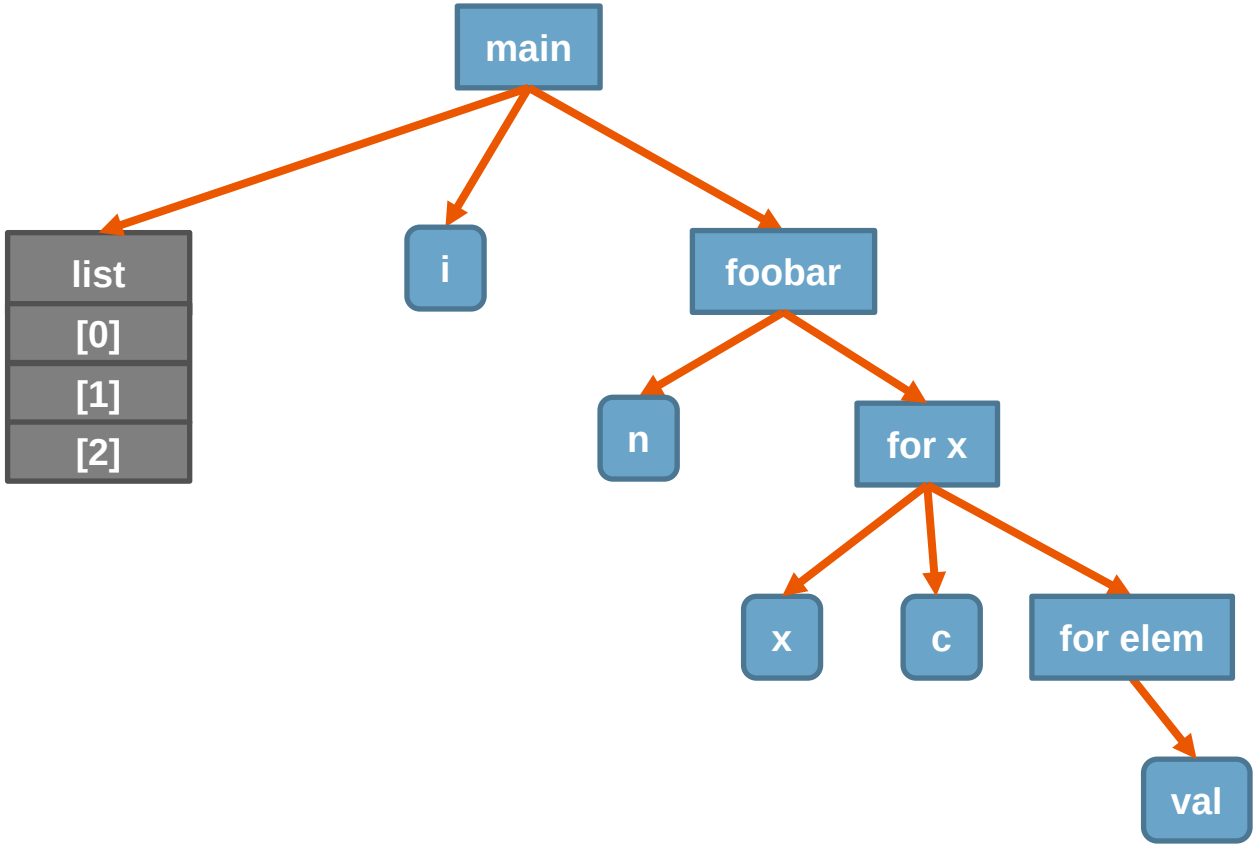



List elements pointing to string slices
➔ borrow

## *b) Ownership Diagram:*

```
fn foobar(n: i32, list: &Vec<Option<& str>>) {
    for x in {0..n} {
        let mut c = 0;
        for elem in list {
            let val = elem.unwrap();
            println!("{}.{}: {}", x, c, val);
            c += 1;
        }
    }
}
```




main
list
[0]
[1]
[2]
i
foobar
n
for x
x
c
for elem
val

# Exercise 5 – Q4 - a:

***a) Write a  function to sum the area of a list of Shapes. Assume constant PI is defined.***

```
enum Shape {
    Rectangle(f64, f64), /* width, height */
    Triangle(f64, f64), /* height, base */
    Circle(f64), /* radius */
}
```

```
enum Shape {
    Rectangle(f64, f64), /* width, height */
    Triangle(f64, f64), /* height, base */
    Circle(f64), /* radius */
}
```

```
fn total_area(list: &Vec<Shape>) -> f64 {
    let mut sum = 0.0;
    for s in list {
        sum += match s {
            Rectangle(w, h) => w * h,
            Triangle(b, h) => b * h / 2.,
            Circle(r) => PI * r * r,
        };
    }
    sum
}
```

# Exercise 5 – Q4 - b:

***b) parse a csv file filled with integers and return a 2-dimensional vector of integers.***

*Hint: you can parse a string to an integer using the parse method. Return an **io::Error with ErrorKind::Other** if a value cannot be parsed to an integer.*

```rust
let mut f = File::open(filename)?;
    let mut s = String::new();
    f.read_to_string(&mut s)?;

    let mut ret = Vec::new();
    let rows = s.split("\n");
    for row in rows {
        if row.trim().len() == 0 {
            continue;
        }

        let cols = row.split(",");
        let mut retrow = Vec::new();
        for col in cols {
            match col.trim().parse::<isize>() {
                Ok(val) => retrow.push(val),
                Err(e) => return Err(Error::new(ErrorKind::Other,
                                format!("{}", e))),
            }
        }

        ret.push(retrow);
    }

    Ok(ret)
```

```rust
fn parse_csv(filename: &str) ->
        Result<Vec<Vec<isize>>, io::Error>
```

# Exercise 5 – Q4 - c:

***c) Write a function, reverse_vector, that takes an input vector of integers and will return an output vector whose elements are reversed. You can only use recursion to solve this problem.***

*Hint: you will need a helper function to do this.*

```
fn reverse_vector(v: & Vec<i32>) -> Vec<i32> {

    let s = &v[..];                          // turn v into a slice
    return reverse_vector_helper(s);         // recursion starts
}

fn reverse_vector_helper(v: & [i32]) -> Vec<i32> {
    if v.len() == 0 {
        return Vec::new();
    }

    let i = v[0];
    let v = &v[1..];
    let mut ret = reverse_vector_helper(v);

    ret.push(i);
    return ret;
}
```

(Code Example)

Thanks for listening!