ECE522 Assignment 7

Zhaoyi Wang 1689747

Question 1

For Question a)

We will define our `transfer()` function like the following:

```
fn transfer(&self, from: usize, to: usize, amount: i32) -> Result<(), ()> {
        let mut temp = self.accounts.lock().unwrap();
        let mut flag = false;
        if temp.contains(&(from as i32)) && temp.contains(&(to as i32)) {
            flag = true;
        }
        if flag == true {
            println!("Amount of ${} transferred from account id: {} to account id: {}.",
amount, from, to);
            Ok(())
        } else {
            Err(())
        }
    }
```

The output example:

```
fn main() {
    let account = Bank::new(20);
    println!("{:?}",account.transfer(5, 10, 100));
}
```

```
Amount of $100 transferred from account id: 5 to account id: 10.
Ok(())
```

For Question b)

```rust
fn main() {
    let bank_ac = Bank::new(15);
    for i in 0..16{
        let ac = bank_ac.clone();
        let person = Person::new(i, i + 1);
        let handle = thread::spawn(move || {
            ac.transfer(person.ac_id, person.buddy_id, 100);
        });
        handle.join().unwrap();
    }
}
```

For the output:

```
Amount of $100 transferred from account id: 0 to account id: 1.
Amount of $100 transferred from account id: 1 to account id: 2.
Amount of $100 transferred from account id: 2 to account id: 3.
Amount of $100 transferred from account id: 3 to account id: 4.
Amount of $100 transferred from account id: 4 to account id: 5.
Amount of $100 transferred from account id: 5 to account id: 6.
Amount of $100 transferred from account id: 6 to account id: 7.
Amount of $100 transferred from account id: 7 to account id: 8.
Amount of $100 transferred from account id: 8 to account id: 9.
Amount of $100 transferred from account id: 9 to account id: 10.
Amount of $100 transferred from account id: 10 to account id: 11.
Amount of $100 transferred from account id: 11 to account id: 12.
Amount of $100 transferred from account id: 12 to account id: 13.
Amount of $100 transferred from account id: 13 to account id: 14.
```

Question 2

For Question a)

The reason is:

> `thread::spawn(move || { sample_data[0] += i; });`
>
> Value moved into closure, in previous iteration of loop.

For Question b)

We update the code like the following:

```rust
use std::sync::{Arc, Mutex};

fn main()
{
    let mut sample_data = Arc::new(Mutex::new(vec![1, 81, 107]));
    for i in 0..10
    {
        let data = sample_data.clone();
        thread::spawn(move || { data.lock().unwrap()[0] += i; });
    }
    thread::sleep(Duration::from_millis(50));
    println!("{:?}", sample_data);
}
```

For the output

```
Mutex { data: [46, 81, 107], poisoned: false, .. }
```

Question 3

For our `find_words()` function:

```rust
fn find_words(quote: String, ch: char) {
    let words: Vec<_> = quote.split_whitespace().collect();
    let words_with_ch: Vec<_> = words.par_iter().filter(|word| word.contains(ch)).collect();
    println!("The following words contain the letter {:?}: {:?}", ch, words_with_ch);
}
```

For the output:

```
The following words contain the letter 's': ["some", "greatness,", "some", "greatness",
"thrust"]
```

Question 4

```rust
use rayon::prelude::*;

fn concurrent_quick_sort(v: &mut [usize]) {
    if v.len() > 1 {
        let mut mid = partition(v);
        if mid < v.len() / 2 {
            mid += 1;
        }
        let (left, right) = v.split_at_mut(mid);
        rayon::join(|| concurrent_quick_sort(left),
                    || concurrent_quick_sort(right));
    }
}


fn partition(v: &mut [usize]) -> usize {
```

```
    let l = v.len();  // v.len()
    let mut mid = vec![0; l];
    let pivot = v[0];

    let mut less_array = vec![0; l];
    let mut greater_array = vec![0; l];
    let mut equal_array = vec![0; l];

    for i in 0..l {
        mid[i] = v[i];
        if mid[i] < pivot {
            less_array[i] = 1 as usize;
        } else if mid[i] > pivot {
            greater_array[i] = 1 as usize;
        } else if mid[i] == pivot {
            equal_array[i] = 1 as usize;
        }
    }

    less_array = parallel_prefix_sum(&mut less_array);
    greater_array = parallel_prefix_sum(&mut greater_array);
    equal_array = parallel_prefix_sum(&mut equal_array);

    for i in 0..l {
        if mid[i] < pivot {
            v[less_array[i] - 1] = mid[i];
        } else if mid[i] > pivot {
            v[less_array[l - 1] + equal_array[l - 1] + greater_array[i] - 1] = mid[i];
        } else if mid[i] == pivot {
            v[less_array[l - 1] + equal_array[l - 1] - 1] = mid[i];
        }
    }

    return less_array[l - 1] as usize;
}

fn parallel_prefix_sum(v: &mut [usize]) -> Vec<usize> {
    let mut temp_vec = Vec::new();
    for i in 0..v.len() {
        let mut sum = 0;
        for j in 0..i + 1 {
            sum = sum + v[j];
        }
        temp_vec.push(sum);
    }
    temp_vec
}
```

In `fn main()`:

```
fn main() {
    let mut arr = vec![5, 1, 0, 6, 2, 4, 9, 3];
    concurrent_quick_sort(&mut arr);
    println!("{:?}", arr);
}
```

The output:

```
[0, 1, 2, 3, 4, 5, 6, 9]
```