Assignment 7: Concurrency

Question 1: Consider the following code:

```
#[derive(Clone,Debug)]
struct Bank{
 accounts:Arc<Mutex<Vec<i32>>>
}

impl Bank{
   fn new(n:usize)->Self{
      let mut v = Vec::with_capacity(n);
      for _ in 0..n{
            v.push(0);
      }
      Bank{
            accounts:Arc::new(Mutex::new(v)),
      }
 }

   fn transfer(&self, from:usize, to:usize, amount:i32)->Result<(),()>{
      …..
 }
}
```

a- Create a *transfer* function for type "Bank" with the following signature:

pub fn transfer(&self, from:usize, to:usize, amount:i32)->Result<(),()>

The function returns an error if either account [from, to] does not exist. Assuming both accounts exist, the function should transfer the amount and print as follows:

Amount of $33 transferred from account id: 16 to account id: 15.

(Note: The index of the vector "v" represents the account id.)

b- In the main function, you should simulate at least 10 users and make each running on their own thread, going into the bank, making a payment to somebody else's account (i.e. buddy_id). Utilize the following structure modeling a person in your solution:

```
struct Person{
 ac_id:usize,
 buddy_id:usize,
}
impl Person{
 pub fn new(id:usize,b_id:usize)->Self{
      Person{
            ac_id:id,
            buddy_id:b_id
      }
 }
```

```
}
```

Upload your answer as a Rust Project with the name "Bank_Application"

Question 2: Consider the following code:

```
use std::thread;
use std::time::Duration;
fn main()
{
    let mut sample_data = vec![1, 81, 107];
    for i in 0..10
   {
        thread::spawn(move || { sample_data[0] += i; }); // fails here
   }

    thread::sleep(Duration::from_millis(50));
}
```

a- The previous code will not run, explain why (Hint: think about ownership).

b- Apply Mutex to the previous code, so it runs.

Upload your answer as a Rust Project with the name "Assignment7_Q2"

Question 3: Consider the following code:

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

  fn main() {
    let quote = "Some are born great, some achieve greatness, and some have greatness
thrust upon them.".to_string();

   find_words(quote,'s');
  }

  fn find_words(quote:String, ch:char){
       let words: Vec<_> = quote.split_whitespace().collect();
       // add your code here:
       let words_with_a: Vec<_> = ……
       println!("The following words contain the letter {:?}: {:?}", ch, words_with_ch
    );
  }
```

Utilize the rayon crate to iterate through words parallelly and print the words that contain a specific character *ch*.

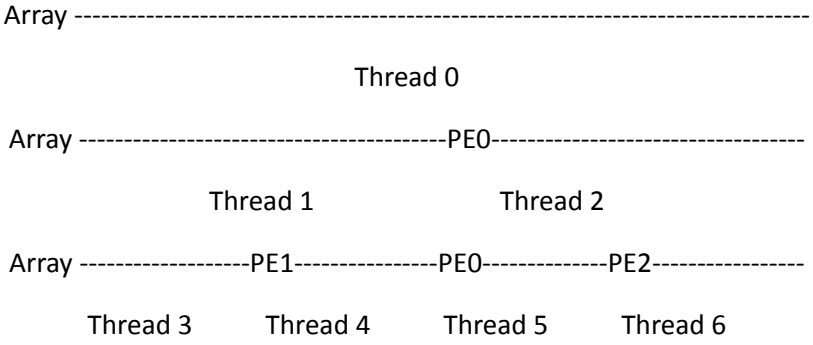
Upload your answer as a Rust Project with the name "Assignment7_Q3".

Question 4: Consider the following code:

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

fn concurrent_quick_sort<T>(v: &mut [T]) {
    //add your code here:
    //uses partition fn, multiple options exist.
    //use rayon for concurrency.
}
fn partition<T>(v: &mut [T]) -> usize {
    //add your code here:
}
```

In Concurrent quick sort, each part is processed by an independent thread (i.e., different threads will find the pivot element in each part recursively). Check following diagram. Here PE stands for Pivot Element.



If you need more information, you can check out this video:
https://www.youtube.com/watch?v=kbkpILBv6fM

The partitioning step is accomplished through the use of a *parallel prefix sum* algorithm (https://en.wikipedia.org/wiki/Prefix_sum) to compute an index for each array element in its section of the partitioned array.

Upload your answer as a Rust Project with the name "Assignment7_Q4".