Quiz 3 Solution

University of Toronto

Duration: 50 minutes
Examiner: Kuei (Jack) Sun

Please fill your student number, last and first name below and then read the instructions carefully.

Student Number: ____ ____ ____ ____ ____ ____ ____ ____ ____ ____

Last Name: ______________________________

First Name: ______________________________

Instructions

Examination Aids: Ruler and examiner approved aid sheet are allowed.

Do not turn this page until you have received the signal to start.

You may remove the aid sheet from the back of this test book. Do not remove any other sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted. Use the blank space in last page as scratch space. Its content will not be marked.

This exam consists of 3 question on 6 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 20 marks.

For the written answers, explain your reasoning clearly. Be as brief and specific as possible. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements in an answer. Please write legibly!

Work independently.

MARKING GUIDE

Q1:	________	(4)
Q2:	________	(7)
Q3:	________	(9)
Total:	________	(20)

Question 1. Positively Odd [4 marks]

Complete the function, `odd`, which takes an array slice of integers, and returns a vector containing only positive odd numbers. You must write this function in one line of Rust code without a semicolon. Hint: it is intentional that the vector contains immutable references to i32 integers.

```
fn odd(v: &[i32]) -> Vec<&i32> {


  // fine to use into_iter()

  v.iter()                                // 1 mark
   .filter(                               // 1 mark
       |&x| *x % 2 != 0 && *x > 0)        // 1 mark
   .collect()                             // 1 mark


  // fine to split into two filter calls
















}
```

Question 2. Spell Caster [7 marks]

You and your friend are designing an action RPG game where the player can cast spells to damage enemy units. Your friend has written the starter code shown here, but you do not like how it is done. Change the starter code so that each spell is a struct that implements the Spell trait (you have to define it yourself). Update the `get_spell` function so that it returns a **trait object** instead. Do not change how the spells work.

```
struct Unit {
    hp: i32,
}

enum Spell {
    Fire(i32),  // damage
    Wind(f64),  // percent damage
    Void,       // do nothing
}

impl Spell {
    fn cast(&self, unit: &mut Unit) {
        unit.hp -= match self {
            Spell::Fire(dmg) => *dmg,
            Spell::Wind(pdg) => (unit.hp as f64 * pdg) as i32,
            Spell::Void => 0,
        };
    }
}

fn get_spell(key: char) -> Spell {
    match key {
        'f' => Spell::Fire(25),
        'w' => Spell::Wind(0.2),
         _  => Spell::Void,
    }
}
```

Note that your solution will be tested with the following code in the main function:

```
let mut target = Unit { hp: 100 };
let spell = get_spell('f'); // can also be 'w' or 'v'
spell.cast(&mut target);
```

```
// SOLUTION

// 1 mark
trait Spell {
    fn cast(&self, unit: &mut Unit);
}

// 1 mark (for all 3 structure definitions)
// Can use regular structs too (e.g., struct Fire { dmg : i32 })
struct Fire(i32);
struct Wind(f64);
struct Void;

// 1 mark
impl Spell for Void {
    fn cast(&self, _: &mut Unit) { }
}

// 1 mark
impl Spell for Fire {
    fn cast(&self, unit: &mut Unit) {
        unit.hp -= self.0;
    }
}

// 1 mark
impl Spell for Wind {
    fn cast(&self, unit: &mut Unit) {
        unit.hp -= (unit.hp as f64 * self.0) as i32;
    }
}

fn get_spell(key: char) -> Box<dyn Spell> {  // 1 mark
    match key {
        // 1 mark for using Box
        'f' => Box::new(Fire(25)),
        'w' => Box::new(Wind(0.2)),
         _  => Box::new(Void),
    }
}
```

Question 3. Parallel Find [9 marks]

Implement the function, `parallel_find`, which takes an integer `x` and a large, unsorted array of unique integers, `arr`, and spawn some number of threads such that each thread will search a partition of the array for `x`, where the size of each partition is either `nelems` or the size of the last partition. For example, if the array size is 250, and the argument `nelems` is 100, then 3 threads will be created. The first thread will search the index from 0 to 99, the second thread will search the index from 100 to 199, and the last thread will search the remaining elements from 200 to 249.

You are required to use `mpsc::channel` to solve this problem. Failure to do so will result in no marks given.

```
fn parallel_find(x: i32, arr: Arc<[i32]>, nelems: usize)
    -> Option<usize> {
    let (tx, rx) = mpsc::channel();
    let mut remain = arr.len();
    let mut offset = 0;

    while remain > 0 {
        // 1 mark
        let psize = if remain > nelems { nelems } else { remain };
        let tx = mpsc::Sender::clone(&tx);     // 1 mark
        let arr = arr.clone();                 // 1 mark

        thread::spawn(move || {                // 1 mark
           for i in offset..offset+psize {     // 1 mark
               if arr[i] == x {
                   return tx.send(i).unwrap(); // 1 mark
               }
           }
        });

        remain -= psize;
        offset += psize;
    }

    drop(tx);                   // 1 mark
    for idx in rx {             // 1 mark
        return Some(idx);       // 1 mark
    }
    return None;
}
```

[*Use the space below for rough work*]

END OF EXAMINATION