High Performance Rust MSc Questionnaire
Jim Walker
s1893750@ed.ac.uk

Questionnaire Information

About this project:
This project aims to evaluate the usability of Rust from the perspective of HPC programmers.

Who is responsible for data collected?
Jim Walker

What is involved in this study?
A multiple choice paper questionnaires which asks participants what particular fragments of Rust code do. Participants are also requested to self identify how proficient they are at the following programming languages: Fortran, C, C++, Python, Ruby, Java, JavaScript, Haskell and Rust. I will collect no other data from the participants.

Once questionnaires have been completed I will digitise the data into a csv format, and use it to create figures for my dissertation. Once my dissertation has been marked, that data will be destroyed. The digital csv file holding all my data will only be available on my laptop.

What are the risks involved in this study?
Data could be taken from my laptop, and people could try to ascertain which participants got higher marks on the questionnaire from the skill levels the participant applied to the various languages. This could then potentially effect a participants future career progress if the Rust programming language begun to be widely used in EPCC.

What are the benefits of taking part in this study?
People can test their knowledge on Rust. Once all data has been collected, correct answers will be circulated through the EPCC mailing list.

What are your rights as a participant?
Taking part in the study is voluntary. You may choose not to take part or subsequently cease participation at any time.

Will I receive any payment or monetary benefits?
No.

For more information:
You can contact Jim Walker directly, or his supervisor Magnus Morton, m.morton@epcc.ed.ac.uk

Question 1

What does the function `foo` do?

```
fn foo(m: i32, n: i32) -> i32 {
    if m == 0 {
        n.abs()
    } else {
        foo(n % m, m)
    }
}
```

☐ It finds the greatest common divisor of m and n

☐ It doesn't compile.

☐ It finds the closest prime number to n

☐ It calls itself infinitely.

Question 2

In Rust, `vec!` is used to create a vector. All variables in Rust are immutable by default. What happens when we try to run this program?

```
let v = vec![2,3];
v.push(3);
print!("{:?}", v);
```

☐ [2,3,2] is printed.

☐ [2,2,2,3] is printed.

☐ The program does not compile.

☐ The program compiles, but crashes when it tries to push 3 to v.

Question 3

Idomatic Rust code oten uses patterns associated with functional languages. Given an immutable vector, v, please select what the line of code below does.

```
let a = v.iter().fold(1, |acc, x| acc * x);
```

☐ Every element of v is set to 1, and then copied to a.

☐ Every element of v is multiplied together and the result is stored in a.

☐ Every element of v is multiplied by 1 and the result is stored in a.

☐ The program does not compile.

Question 4

A vector's push method return an optional value, or none. What does this fragment of code print?

```
let mut stack = Vec::new();

stack.push(1);
stack.push(2);
stack.push(3);

while let Some(top) = stack.pop() {
  print!("{} ", top);
}
```

☐ Some(3) Some(2) Some(1)

☐ 3 2 1 None None None...

☐ 3 2 1

☐ Some(3) Some(2) Some(1) None None None...

Question 5

What does this fragment of code do?

```
let a: Vec<i32> = (1..).step_by(3)
                       .take(3)
                       .map(|x| x * 2)
                       .collect();
```

☐ Sets a to [2, 4, 6]

☐ The program doesn't compile.

☐ [4, 10, 16]

☐ [2. 8, 14]

Question 6

In this question, a and b are both vectors of the same length. The method `par_chunks` returns a parallel iterator over at most `chunk_size` elements at a time. What does this fragment of code do?

```
a.par_chunks(chunk_size)
    .zip(b.par_chunks(chunk_size))
    .map(|(x,y)| x.iter()
                  .zip(y.iter())
                  .fold(0, |acc, ele| acc + *ele.0 * *ele.1)
    ).sum();
```

☐ Sum reduction

☐ Dot Product

☐ Element wise sum

Question 7

The Rust compiler's borrow checker makes sure that values are mutably borrowed if they are altered from a different function than the one they were created in. What does this program do?

```
fn plus_one(x: &mut i32){
    *x += 1;
}
fn main(){
    let x = 64;
    plus_one(&mut x);
    println!("{}", x+1);
}
```

☐ Print 65.

☐ Prints an undefined value.

☐ It doesn't compile.

☐ Print 66.

Question 8

Please tick the boxes below to show your level of skill in the varying programming languages

	None	Basic	Comprehensive	Advanced
Fortran				
C				
C++				
Python				
Ruby				
Java				
JavaScript				
Haskell				
Rust				