

Theory exercises

Program verification with types and logic (NWI-IMC060)

Week 7

1. This question is about programming in Rust.

- (a) Consider the type `&mut Option<T>`. Give the values of this type and describe the layout of these values in memory. You should describe the layout in memory by drawing a figure showing the pointer structure.
- (b) Consider the following function:

```
pub fn take1<T> (o : &mut Option<T>) -> Option<T> {  
    match *o {  
        None => None,  
        Some(x) => Some(x)  
    }  
}
```

Is the function `take1` safe?

- If not, give a client of the function that causes a memory unsafety.
- If yes, explain why the function is safe and if the Rust type system accepts it.

(c) Consider the following function:

```
pub fn take2<T> (o : &mut Option<T>) -> Option<T> {  
    match *o {  
        None => None,  
        Some(x) => { *o = None; Some(x) }  
    }  
}
```

Is the function `take2` safe?

- If not, give a client of the function that causes a memory unsafety.
- If yes, explain why the function is safe and if the Rust type system accepts it.

2. This question is about programming in Rust.

- (a) Explain the difference between Rust types that are `Clone` and `Copy`. Your answer should make the following clear:
- Is every type that is `Clone` also `Copy`. If yes, explain why. If not, give a counterexample.
 - Is every type that is `Copy` also `Clone`. If yes, explain why. If not, give a counterexample.

(b) Consider the following function:

```
fn swap1(v : &mut Vec<i32>, i : usize, j : usize) {  
    let x = v[i];  
    v[i] = v[j];  
    v[j] = x  
}
```

(Here, `usize` is a large integer type that can represent all array indices.)

Does the Rust type system accept the function `swap1`? If yes, explain why. If not, explain exactly where the Rust type checker will complain.

(c) Implement the function:

```
fn swap2<T : Clone>(v : &mut Vec<T>, i : usize, j : usize)
```

You should explain why the Rust type checker accepts your implementation.

3. This question is about programming in Rust.

(a) We have the type `Vec<i32>`, the type `&Vec<i32>`, and the type `&mut Vec<i32>`. Explain how these types are represented in memory.

(b) Briefly explain which of the following operations you can do with the underlying vector via each of the types `Vec<i32>`, the type `&Vec<i32>`, and the type `&mut Vec<i32>`:

1. Deallocate the vector and its elements.
2. Mutate the vector.
3. Obtain multiple aliases to the vector.

(c) Explain if the following program is accepted by the Rust compiler:

```
let v = vec![1,2,3];  
let w = &v[1];  
v.push(4);  
println!("{}", *w);
```

If yes, explain why. If no, explain what the Rust compiler will say. Also explain whether or not the program is safe.