Challenges 1.4

Challenge 1.4.1

Implement, as best as you can, the identity function in your favorite language (or the second favorite, if your favorite language happens to be Haskell).
Solution for Rust:

```
pub fn my_id<T>(a:T)->T{a}

```

Challenge 1.4.2

Implement the composition function in your favorite language. It takes two functions as arguments and returns a function that is their composition.
Solution for Rust:

```
pub fn compose<'a,F,G,A,B,C>(f:F, g:G) -> Box<(dyn Fn(A)->C + 'a)>
where
F: Fn(A)->B + 'a,
G: Fn(B)->C + 'a,
{
    return Box::new(move |a| g(f(a)) );
}

```

Challenge 1.4.3

Write a program that tries to test that your composition function respects identity.
This isn't possible. With Rice's theorem it follows, that such questions for Programs are undecidable.

Challenge 1.4.4

Is the world-wide web a category in any sense? Are links morphisms?
With links as morphisms the world-wide web isn't a category. This is because wo don't have composition for links. If website A has a link to website B and B a link to website C there isn't nesecery a link from A to C.

Challenge 1.4.5

Is Facebook a category, with people as objects and friendships as morphisms?
No because we don't have composition. if A and B ara friends and B and C are friends A and C don't have to be friends.

Challenge 1.4.6

When is a directed graph a category?
If it is closed under reflexivity and transitivity.

Challenges 2.7

Challenge 2.7.1

Define a higher-order function (or a function object) memoize in your favorite language. This function takes a pure function f as an argument and returns a function that behaves almost exactly like f, except that it only calls the original function once for every argument, stores the result internally, and subsequently returns this stored result every time it's called with the same argument. You can tell the memoized function from the original by watching its performance. For instance, try to memoize a function that takes a long time to evaluate. You'll have to wait for the result the first time you call it, but on subsequent calls, with the same argument, you should get the result immediately.

```
use std::{collections::HashMap, hash::Hash};

struct Memoized<F,A,B>
where
    F: Fn(A)->B,
    A: Clone + Hash + Eq,
    B: Clone
{
    function : F,
    memory: HashMap<A,B>
}

impl<F,A,B> Memoized<F,A,B>
where
```

```
    F: Fn(A)->B,
    A: Clone + Hash + Eq,
    B: Clone
{
    fn new(f:F) -> Memoized<F,A,B>{
        Memoized {
            function: f,
            memory: HashMap::new(),
        }
    }

    fn apply(&self, arg:A)->B {
        if self.memory.contains_key(&arg) {
            return self.memory.get(&arg).unwrap().clone();
        } else {
            let result = (self.function)(arg);
            self.memory.insert(arg.clone(), result.clone());
            return result;
        }

    }
}

```

Challenge 2.7.2

Try to memoize a function from your standard library that you normally use to produce random numbers. Does it work?
This wouldn't work, because a random number generator isn't a pure function. If memoized it would generate the first number random but then it would always return exactly this memoized random number.

Challenge 2.7.3

Most random number generators can be initialized with a seed. Implement a function that takes a seed, calls the random number generator with that seed, and returns the result. Memoize that function. Does it work?
This would work, because now we just memoize the seed for which a fixed random number generator is generated. The calls to the random number generator are not memoized, so they produce a new random number every

time the function is called. The only artifact ist, that if we try to get a new random number generator with the same seed, we don't realy get a new one. We get the one that already was instantiated and therefore we don't get the random number sequence from the start but from where it is currently located.

Challenge 2.7.4

Which of these C++ functions are pure? Try to memoize them and observe what happens when you call them multiple times: memoized and not.

(a): pure

(b): impure, because it depends ond what the user types, so it gives a different value depending on user input not on function input

(c): pure in the sense of the return behaviour, but it has an effect an is therefore impure in this regard.

(d): impure because the static int y variable ist an inner state of the function. If we call it f(1) it returns 1 if we then call it again with the same argument f(1) it returns 2

Challenge 2.7.5

How many different functions are there from Bool to Bool? Can you implement them all?
There are 4 different functions because the function can have two different values for a True input and two different values for a False input. All the function are

- f1 b = if b then True else True
- f2 b = if b then True else False
- f3 b = if b then False else True
- f4 b = if b then False else False

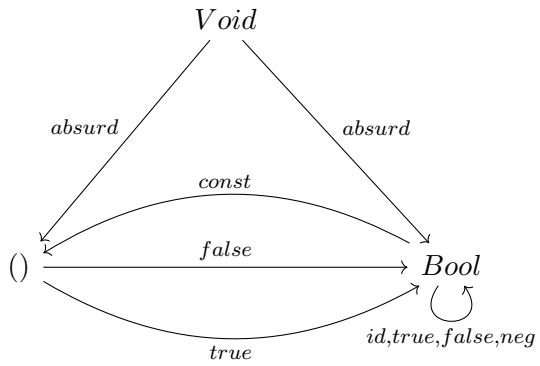

Figure 1: Category for Challenge 2.7.6

Challenge 2.7.6

Draw a picture of a category whose only objects are the types Void, () (unit), and Bool; with arrows corresponding to all possible functions between these types. Label the arrows with the names of the functions.

Challenges 3.6

Challenge 3.6.1

Generate a free category from:

(a) A graph with one node and no edges
The Category just consists of the one node as an object, and the added identity arrow.

(b) graph with one node and one (directed) edge (hint: this edge can be composed with itself)
The Category also has just the one node as an object with an added identity arrow. This time though we also add the arrows for arbitrary sequences of composing the one initial edge. So if e is our edge we get an arrow for e^n for $n \in \mathcal{N}$

(c) A graph with two nodes and a single arrow between them
The category just consists of the two nodes as objects with added identity arrows and the one arrow between them. There is no pair of arrows we can

compose besides those with identity arrows, so we don't generate any new arrows.

(d) A graph with a single node and 26 arrows marked with the letters of the alphabet: a, b, c ... z.
The category consists of the one node as an object and infinitly many arrows labeled with all the possible strings over the alphabet. So for every $w \in \{a, ..., z\}^+$ we have an arrow. The Category can be seen as the category of string concatination. The identity arrow then would be the same as a ϵ arrow

Challenge 3.6.2

What kind of order is this?

(a) A set of sets with the inclusion relation: A is included in B if every element of A is also an element of B.
It's a partial order, its reflexive, transitive, and antisymetric. Also not all Sets have to be comparable for example neither $\{a\} \subseteq \{b\}$ nor $\{b\} \subseteq \{a\}$ holds

(b) C++ types with the following subtyping relation: T1 is a subtype of T2 if a pointer to T1 can be passed to a function that expects a pointer to T2 without triggering a compilation error.
???

Challenge 3.6.3

Considering that Bool is a set of two values True and False, show that it forms two (set-theoretical) monoids with respect to, respectively, operator && (AND) and || (OR).
&& and || are associative and they always yield a Bool. For && the neutral element is 1 because $1\&\&x = x$ and $x\&\&1 = x$ and for || it is 0 because $0||x = x$ and $x||0 = 0$

Challenge 3.6.4

Represent the Bool monoid with the AND operator as a category: List the morphisms and their rules of composition.
The id morphism is *true* and the composition is $f \circ g = f\&\&g$

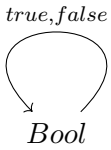

Figure 2: Category for Challenge 3.6.4

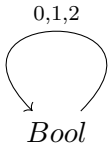

Figure 3: Category for Challenge 3.6.5

Challenge 3.6.5

Represent addition modulo 3 as a monoid category.
The id morphism is 1 and the composition is $f \circ g = (f + g) \bmod 3$

Challenges 5.8

Challenge 5.8.1

Show that the terminal object is unique up to unique isomorphism.