Advanced Programming
3rd Assignment: Advanced Type Systems
Deadline: 19 December 2025, 23:59

Programming Group
guido.salvaneschi@unisg.ch

University of St.Gallen

1 Assignment

In December 1998, NASA launched its Mars Climate Orbiter, which cost around 300 million USD. Nine months later, the orbiter arrived in the orbit of Mars where it was supposed to remain. However, it approached the planet too closely and disintegrated in its atmosphere. What caused this mistake? A mismatch in units. The two teams working on this project used different systems: NASA used metric and Lockheed Martin used imperial.
In this assignment, we will use Typescript's advanced types to develop a library of scientific operations on distances and times, which will produce compile-time errors if units aren't as expected. After you have completed the assignment, please submit your solution using the course website. You should submit ONLY one file, called `assignment.ts`.

1.1 Quantities

We will represent quantities with objects. Distances will look like:

```
{ value: 10, unit: "meter" }
```

while times will look like

```
{ value: 20, unit: "second" }
```

Task 1: Values, Distances, and Times

Let's assume for now that distances are always in meters and times are always in seconds. Define the type function `Value`, which represents a value with a unit (like above), and use it to define the types `DistanceMeters` and `TimeSeconds`.

Task 2: More Units

How would you modify your definitions of `DistanceMeters` and `TimeSeconds` to allow kilometers, feet, and yards as distances, and hours and minutes as times? Write the types `DistanceUnits` and `TimeUnits` of the allowed units, then the new general definitions of distance and time into two new types `Distance` and `Time`.

1.2 Addition of Quantities

Let's implement a type-safe addition of two quantities, incrementally.

Task 3: First try

Assume values in a unit `U` have the type `Value<U>`. What is wrong with this definition of `add`?

```
function add<U>(q0: Value<U>, q1: Value<U>): Value<U> {
    return { value: q0.value + q1.value, unit: q0.unit };
}
```

Give an example that type-checks when it should not.

Task 4: Second try

With the same assumption on the type of values, what is wrong with this definition of `add`?

```
function add<U1, U2 extends U1>(q0: Value<U1>, q1: Value<U2>) : Value<U1> {
    return { value: q0.value + q1.value, unit: q0.unit };
}
```

Give an example that type-checks when it should not.

Task 5: Type-Level Equality

Define a type-level function `Same`, that takes two types and returns `never` if they are not equal, and one of the types if they are.

Task 6: Third time's the charm

Using the type-level function defined earlier and intersection types, define a correct `add` function, that only works if it is given values with the same unit.

1.3 Unit Conversion

Now, when adding two quantities they must be in the same unit. But, what if they aren't? We need a way to convert between units.

Task 7: Converting Units

Write two functions `convertDistance` and `convertTime`, one that converts distances and the other that converts times. Demonstrate the former by adding 10 meters to 20 yards and reporting the result in feet.

1.4 Product of Quantities

Finally, we want to be able to multiply two quantities. This is useful, for example, to compute areas. However, multiplying two quantities is more complicated than adding them, because the resulting unit is different from the input units. In fact, the units are also themselves multiplied.

Task 8: Product of Units

Define a type function `Prod` that represents the product between two units.

Task 9: Product of Quantities

Define a `mult` function that multiplies two quantities.

1.5 Adapting to New Requirements

Congratulations! Your library is complete and NASA decided to adopt it. However, their new policy is to use only the following metric units: millimeters, centimeters, meters, and kilometers. Your library is still valuable though, because they still want to avoid adding meters to kilometers! Their regulations also include using libraries as-is. So they are not allowed to modify any of your previous code.

Task 10: Restricting to Metric

Define the metrics from the new requirements as a new type `NewDistanceUnits`, that must use your previous definition of `DistanceUnits` as a black box, i.e. you are not allowed to look at its definition, nor to change it.