National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2018/2019

Mission 12
Package Integration

Release date: 26 October 2018
Due: 03 November 2018, 23:59

Required Files

- mission12-template.py
- generic_arith_min.py

Information

In this mission, we will investigate how we can handle the interactions between different data types.

You will require the code from both Missions 11a and 11b for this mission. Since you have only completed one mission (e.g 11a), you can obtain the code for the other mission (11b) from another student who has done it. Please state clearly in the solution who you worked with. Alternatively, you can just complete both mission 11a and 11b yourself before attempting this mission.

For this mission, you and your partner will be working individually and submitting your own solutions.

This mission has **three** tasks with each task divided into **two** parts.

Operations Across Different Types of Numbers

At this point, all the methods installed in our system require all operands to have the same subtype. There are no methods installed for operations combining operands with distinct subtypes. For example, `is_equal(n3, r3_1)` will return a "no method" error message because there is no equality method at the subtypes (`ordinary, rational`). The system is not currently able to handle interactions between the ordinary number 3 and the rational number 3/1.

Some operations across distinct subtypes are straightforward. For example, to combine a generic rational number with a generic ordinary number, n, we need to *cast* n into the generic rational number $n/1$ and combine them into a generic rational number.

In computer science, type conversion or *typecasting* refers to changing an entity of one datatype into another. Explicit typecasting is know simply as *casting* while the term for implicit typecasting is *coercion*. The functions you will install in Task 3 are said to carry out coercion because they take in arguments of explicitly different types and do type conversions "behind-the-scenes".

Task 1: Converting Types (4 marks)

Define this function in the generic rational number package:

$$\texttt{repord\_to\_reprat} : \text{RepOrd} \rightarrow \text{RepRat}$$

that takes in a RepOrd n and *casts* it into a RepRat, which has a numerator that is the Generic-Ord form of n and a denominator that is the Generic-Ord form of 1. The function should return the new RepRat.

Note: Refer to definitions in Mission 11 for the difference between RepOrd and Generic-Ord, RepRat and Generic-Rat.

Do the same for the generic complex number package. Define a function in the generic complex number package:

$$\texttt{repord\_to\_repcom} : \text{RepOrd} \rightarrow \text{RepCom}$$

that takes in a RepOrd n and casts it into a RepCom whose real part is the Generic-Ord form of n and whose imaginary part is the Generic-Ord form of 0. The function should return the new RepCom.

Note: Refer to definitions in Mission 11 for the difference between RepOrd and Generic-Ord, RepCom and Generic-Com.

Task 2: Coercive Functions (3 marks)

The function `RRmethod_to_ORmethod`[1] makes it possible to obtain a $(\text{RepOrd}, \text{RepRat}) \rightarrow T$ method from a $(\text{RepRat}, \text{RepRat}) \rightarrow T$ method, for *any* type T, as shown below:

```
def RRmethod_to_ORmethod(method):
    return lambda ord, rat: method(repord_to_reprat(ord), rat)
```

For example, `RRmethod_to_ORmethod(add_rat)` would allow `add_rat` to add a RepOrd *ord* to a RepRat *rat* after the relevant arguments have been provided.

Define the corresponding function `RRmethod_to_ROmethod` that for any type T can be used to obtain a $(\text{RepRat}, \text{RepOrd}) \rightarrow T$ method from a $(\text{RepRat}, \text{RepRat}) \rightarrow T$ method.

For the generic complex number package, define similar functions `CCmethod_to_OCmethod` and `CCmethod_to_COmethod` which will allow us to obtain a $(\text{RepOrd}, \text{RepCom}) \rightarrow T$ method from a $(\text{RepCom}, \text{RepCom}) \rightarrow T$ method and a $(\text{RepCom}, \text{RepOrd}) \rightarrow T$ method from a $(\text{RepCom}, \text{RepCom}) \rightarrow T$ respectively (for any type T).

[1] R stands for Rat, O stands for Ord

Task 3: Installation (10 marks)

Using the methods `RRmethod_to_ORmethod` and `RRmethod_to_ROmethod`, modify the Rational Number Package to define methods for generic `add`, `sub`, `mul`, `div`, `is_equal` for argument types (`ordinary`, `rational`) and for argument types (`rational`, `ordinary`).

Install your new methods. Test them on `is_equal(n3, r3_1)` and `is_equal(sub(add(n3, r2_7), r2_7), n3)`

Do the same for the generic complex number package.

Using `CCmethod_to_OCmethod` and `CCmethod_to_COmethod`, modify the Complex Number Package to define methods for generic `add`, `sub`, `mul`, `div`, `is_equal` for argument types (`ordinary`, `complex`) and for argument types (`complex`, `ordinary`).

Install your new methods. Test them on `is_equal(n3, c3_plus_0i)` and `is_equal(sub(add(n3, c2_plus_7i), c2_plus_7i), n3)`

(0.5 mark per correct installation)