Programming Lanaugages (4) Parametric Polymorphism (aka Generic Types/Functions)

Kenjiro Taura

Motivation

want to write

- a function that *sorts arrays of various types* (e.g., ints, floats, strings, structs, . . .)
- a function that *extracts elements from a list satisfying* $p(x)$
- *containers including stacks, queues, trees, graphs, hashtables, etc.* of various types, . . .
- variety of *graph algorithms (breadth-first search, depth-first search, connected components, partitioning, etc.)* that can/should work regardless of the exact data type of each node
- . . .

without duplicating code for each underlying type

A trivial example (generic function)

write a function

$$f(a) = a[0]$$

in your language (an element of an array, let's say)
Questions:

- do you have to specify the type of a?
- if so, how you can say *"a must be an array but whose element can be any type"*
- if not, can it automatically apply to any array?
  - *does it type-check statically* (i.e., what if you pass something not an array)?

So that you don't get bogged down ...

things are conceptually straightforward, pains are around *spelling out types*; *just master the syntax*

- a type of *functions taking an integer and returning a float*
  - Go : `func (int64) float64`
  - Julia :
  - OCaml : `int -> float`
  - Rust : `fn (i64) -> f64`
- a type of typical containers, such as array/slice/vector of ints, list of floats, etc.
- *for any type, satisfying an interface/trait, this function* takes a parameter of type (array of T) and returns a value of type (T)