

# Defining a generic scalar-type package in Ada

Asked 7 years, 10 months ago   Active 7 years, 10 months ago   Viewed 903 times



3



I'd like to test the waters of writing Ada packages by making one for manipulating polynomials. Polynomials can be defined for a wide class of algebraic structures, so to reflect this, I'd like to make the package generic, so it can be used with Floats, Integers, or other numeric sub-types.

I want to say now that I know very little about how Ada's type system work or how its package system works. There seems to be a lack of good beginner Ada information on the web, so I'm having to glean what wisdom I can from [this](#) not-so-newbie-friendly Wikibooks article.

[This](#) page has some information about the type hierarchy. Based on that, it would seem that a reasonable type for my Polynomial package to be based on would be the `Scalar` type, since apparently that's the one on which arithmetic operations are defined. So this is what I've attempted, in `polynomials.ads`:

```
generic
    MAX_DEGREE : Positive;
    type Element is new Scalar;

package Polynomial is

    type Polynomial is Array (0 .. MAX_DEGREE) of Element;

end Polynomial;
```

However, this just nets me a "Scalar" is undefined error from GNAT.

So far I've really just been feeling my way around half-blind, I don't actually know how any of this stuff works. If I seem to have any major misconceptions that you think need to be cleared up, please tell me. Probably the easiest would be to provide example `polynomial.ads` and `polynomial.adb` code that I can learn from - just like a definition of the Polynomial type (with generic max-degree and element type) and a simple example function like adding two polynomials, so I can see how generic functions work.

PS: Sort of related, is there a way to define attributes for your user-defined types?

[ada](#)

asked Nov 16 '12 at 13:12



Jack M

3,023 6 28 44

- 1 Just to clarify why Scalar doesn't work; the type hierarchy in your link is a chart of *categories* of types, not the actual types. – [egilhh](#) Nov 16 '12 at 14:27

## 2 Answers

Active	Oldest	Votes
--------	--------	-------



The problem is just that "Scalar" is not the name of a type.

7



Looking at the section "Generic formal types" in that article I can't see one that imposes the exact restriction you want : "any scalar type". Pity... Unless someone has a better idea, I would widen the declaration to:



```
type Element is private;
```



and carry on. This may not be a drawback : see the next section on generic formal subprograms, if you supply your own operators

```
with function "*" (X, Y: Element) return Element;
```

or

```
with function "*" (X, Y: Element) return Element is <>;
```

you can then instantiate the generic for records (complex numbers?) matrices etc if it is meaningful to do so. The "is <>" will use existing functions for types that already have them (Float etc) to simplify the instantiations

(Edit : forgot that scalar includes enumerations, for which polynomials or even multiplication, do not usually make sense! So widening it to "private" may not be such a drawback)

edited Nov 16 '12 at 13:58

answered Nov 16 '12 at 13:41



[Brian Drummond](#)

**16.8k** 1 22 38

[see this question](#) for a more detailed example of how to do this. – [NWS](#) Nov 18 '12 at 21:27



6



When you are defining a generic, what formal type you use defines what operations you have available inside the implementation of the generic. You can always take Brian's option and use the (very nearly) most restrictive ( `is private` , which you can copy, but otherwise could be damn near anything), and then make the user define the routines you need. This is in fact the only way to make a single generic that can perform math on any scalar.



Ada's generic [formal scalar type system](#) is divided into the following rough hierarchy:

- There are discretely ( `<>` ), floating point values ( `digits <>` ), and fixed point values ( `delta<>` ).
- Within the discretely, there are signed integers ( `range <>` ) and modular integers ( `mod <>` ) (enumerated types are also discretely, but they can only be handled as discretely).

An important implication of this is that there is a way to make a generic that can take both integers and enumerations ("discretely"), but there is no way to make a generic that can operate on both integers and floats. Instead you either have to make one for each, or fake it out with privates and passed in math operators, like Brian's answer showed.

In practice I haven't found this too much of a problem. The instances where I want to work with floating-point values tend to be very different than those where I want to work with integers. For instance, in this case, how useful would your "polynomial" generic really be for 32-bit integers? Very few calculated results would be exact integers.

**This is sort of a cultural issue.** Many languages (led by C) consider Integers and Floating-point types to be closely related things, and make them easily interchangeable (sometimes even silently). In Ada, they are two totally different universes, and you should get used to thinking of them that way. It is true that it isn't too big of a pain to convert an integer into a floating-point value, and the floating point types have ways to truncate or round their values to the nearest integer. For the most part though, you should keep your integers (discretely) and your floating-point values separated.

So for me, I'd just define it with

```
type Element is digits <>;
```

...and go on my merry way.

edited Nov 16 '12 at 15:22

answered Nov 16 '12 at 15:10




**T.E.D.**

40.3k 8 62 130

- I want to up-vote the para starting "In practice I haven't found this ... a problem". Except for one thing : I am doing a bit of HW/SW cosimulation (imagine DSP code running REALLY FAST on an FPGA because everything happens in parallel). It's common to use fixed-point there, and it would be really useful to unify fixed and float versions of the same algorithm (to investigate rounding errors from different word widths) There is a HUGE market for this (System-C and System-Verilog plus...) that Ada is ideal for (and the more so because of her VHDL cousin) – [Brian Drummond](#) Nov 16 '12 at 15:25

@BrianDurmond - Now, wanting to unify *fixed* and float I kind of whistled past. That I could see a case being made for. Perhaps this would be a good thing to agitate for in the next language revision. In the meantime, yeah, do it the way Brian's answer goes into. – [T.E.D.](#) Nov 16 '12 at 15:40

- 3 ...something like `type Element is digits <> or delta <>;` None of the fixed or float-specific attributes would be available to `Element` within the generic, but all the common math operators would be. :-)
- [T.E.D.](#) Nov 16 '12 at 15:53 
- 
-