Functors

11-15-2004

Opening Discussion

- How do we create modules in ML? How does this compare to other languages you know? What can you see as some downfalls of this method?
- How are you coming on assignment #7? I have now posted a complete description of assignment #8 which is due a week from today.

Revisiting the Binary Tree

- Last time we took our binary tree code and put it in a structure. This had the advantage of allowing us to not pollute the global namespace with the function names. By specifying a certain a signature we were also able to hide the deleteMin function.
- There is still a general problem with our tree, the fact that we have to provide a “less than” function in every call. One way to fix this is to use an outside definition.

The Ideal : A Tree for each Type

- Notice that if we make the tree use an outside function definition, the types of the functions change to match the definition of that function. This way we can get new definitions for each definition of “less than”, but in general having code that depends on outside functions is ugly and the type for deleteMin isn't exactly pleasing either.
- We'd like some nice way of defining new binary tree structures for each comparator without copying the code.

The Solution : Functors

- The way of doing this in ML is with functors. A functor in ML is basically a mechanism for creating new structures in ML. (This is not the same as a functor in C++/Java which is a class wrapping a single function.)
- In its simplest form, a functor takes a structure and produces another structure. The form of the functor is
  - functor <ident> (<struct name>:<signature>) = <structure definition>
- The structure definition is exactly what we did last class.

How to use a Functor

- The signature in the functor defines what type of structure we want passed in. This can be defined before the functor or in the functor declaration.
- We then have to define a structure that fits that signature and “pass” it into the functor.
- The syntax for applying the functor to a structure is as follows.
  - structure <ident> = <functor name>(<structure argument>)
- This way we can create a new structure for each type or comparator.

True Power of Functors

- In general, you can provide the functor with multiple declaration types. These include structures, values, functions, and exceptions. In that situation they should be preceded by the declaration keyword and similar types can be put together with and or all types can be separated by semicolons.
- The use of functors in very much like a template class in C++ other than the fact that structures aren't instantiated the same way and can hold more.

Code

- Make our BST be a functor so that we can create them to work with any comparison function.

Minute Essay

- Functors can give you remarkable flexibility in producing code that works with different types. It has a feel somewhat like dynamic binding in OO languages, but it is still static. How could you use a functor to help make your assignment code more flexible for assignment #8?