λ 's on the edge

Functional Programming in C++ part deux

Alfons Haffmans

April 28, 2013

Introduction

This is a continuation of a previous article in which I discussed support for fucntional programming built into C++. In this article I'll be looking to extend C++ to support advanced functional concepts like functors, applicative functors and monads.

Currying

Currying turns any function into a higher order function of one variable [**?**]. The curry of the function returns a partially applied version of the original.

The operator $curry2$ is a higher order function which takes a binary function as input and returns a unary higher order function. That function is the curry of

the binary function.

$$curry2 \;::\; ((a,b) \to c) \;\; \to \;\; (a \to b \to c)$$
$$f :: (a,b) \to c \;\; \Rightarrow \;\; (curry2\ f) :: a \to b \to c$$

When the curried version of f is called with an argument of type a it returns another unary function Calling this function with an argument of type b returns the same value as f.

$$plus \;::(int\ x, int\ y) \;::int = x + y \;\; \Rightarrow \;\; cplus(int\ x) \;::\; (int \to int)$$
$$\to (int\ y) \;::\; int \to x + y$$
$$plus(5,6) = 11 \;\; \Leftrightarrow \;\; (curry2\ plus)(5)(6) = 11$$

$(curry2plus)$ is the curried version of $plus$. *curry2 plus)(5)* returns a lambda which represents the $plus$ function partially applied to 5. When this partially applied version of $plus$ is called with 6 an unsurprising 11 is the result.

Listing **??** shows an implementation of curry2 in C++:

```
1 template <typename R, typename T, typename U>
2 std::function<std::function<R (U)> (T)> curry(std::function<R (T
      ,U)> op)
3 {
4   return [=] (T x) { return [=] (U y) {return op(x, y);};};
5 }
6   auto l = curry<int,int, int> ([](int x, int y) { return (5 + x
        ) * y;});
```

```
7   std::cout << l(1)(1) << std::endl; //prints 6
```

Listing 1: curry for binary operators

Currying plays an important role in functional programming [**?**]. It simplifies the design of higher order functions because we only have to consider unary functions. C++ does not provide a curry operator and functions are not written in curried form. Compare this to Haskell where functions are curried by default [**?**, **?**]. However writing a curry operator or writing curried versions of a function has become a lot easier now that lambda's are supported.

Functors

Functors generalize the concept of mapping a function over values in a container. Their typeclass is :

class Functor f where

$fmap :: (a \to b) -> f\ a \to f\ b$

In C++ this is defined in the template

```
1 template <template<typename T1, typename... D> class F>
2 struct functor {
3
4   //curried version
5   template<typename A, typename B>
```

```
6   static std::function < F<B> (F<A>)> fmap(std::function <B (A)>
        f);
7
8   // uncurried, for functions..
9   template<typename A, typename B>
10  static F<B> fmap(std::function <B (A)> f, F<A> L) {
11    return fmap(f)(L);
12  }
13 };
```

The class F can have more than one template parameter, as indicated by the variadic template. This allows specialization for containers since they have more than one template parameter.

Useful functors satisfy the following laws

$$fmap\ id = id$$

$$fmap(g\ .\ f) = (fmapf).(fmapf)$$

. is the function composition operattor, read as $after$. It applies the function g after the fucntion f, viz.

$$(g\ .\ f)(x) = g(f(x))$$

0.1 Lists

For lists like $std :: list$ and $std :: foward\_list$ fmap corresponds to map.

Applicative Functors

Monads

Do-like Notation

Conclusions

References

[1] Bjarne Stroustrup

The C++ Programming Language

Addison-Wesley, 1997, 3rd edition.

[2] Brian McNamara, Yannis Smaragdakis

Functional programming with the FC++ library.

J. Funct. Program. 14(4): 429-472 (2004)

[3] David Vandevoorde, Nicolai M. Josuttis

C++ Templates

Addison-Wesley, 2003.

[4] Andrei Alexandrescu

Modern C++ Design

Addison-Wesley, 2001

[5] Miran Lipovača

Learn you a Haskell for great good : a beginner's guide

no starch press, San Fransisco, 2011

[6] Graham Hutton

Programming in Haskell

Cambridge University Press, 2007

[7] Richard Bird *Introduction to Functional Programming using Haskell* Prentice Hall Europe, 1998, 2nd edition

[8] Anthony J. Field and Peter G. Harrison

Functional Programming

Addison-Wesley, 1989.

[9] Michael L. Scott

Programming Language Pragmatics

Morgan Kauffmann, 2006, 2nd edition

[10] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns : Elements of Reusable Object-Oriented Software

Addison Wesley Longman, 1995

[11] Nocolai M. Josuttis

The C++ Standard Library

Addison-Wesley, 2nd edition.

[12] *https://github.com/fons/functional-cpp*

[13] *http://www.macports.org/*

[14] *http://en.cppreference.com/w/cpp/language/lambda*

[15] *http://en.cppreference.com/w/cpp/utility/functional/function*

[16] *http://en.cppreference.com/w/cpp/language/auto*

[17] *http://en.cppreference.com/w/cpp/utility/functional/bind*

[18] *http://en.cppreference.com/w/cpp/utility/functional/placeholders*

[19] *http://en.cppreference.com/w/cpp/algorithm/for_each*

[20] *http://en.cppreference.com/w/cpp/algorithm/forward_list*

[21] *http://en.cppreference.com/w/cpp/algorithm/transform*

[22] *http://en.cppreference.com/w/cpp/language/decltype*

[23] *http://en.cppreference.com/w/cpp/algorithm/accumulate*

[24] zip function in Python

http://docs.python.org/2/library/functions.html#zip

zip function in Ruby

http://ruby-doc.org/core-2.0/Array.html#method-i-zip

Support in Perl

http://search.cpan.org/ lbrocard/Language-Functional-0.05/Functional.pm

[25] Brent Yorgey

The Typeclassopedia The Monad.Reader, Issue 13; p17; 12 March 2009 *www.haskell.org/wikiupload/8/85/TMR-Issue13.pdf*

[26] *http://en.cppreference.com/w/cpp/utility/tuple*