Functores

Problemas

Juan Magallón – Adolfo Muñoz

Grado en Ingeniería Informática

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

2022-03-31

Problema - Functor

Problema - Functor

En un lenguaje orientado a objetos diseña un tipo de dato **`functor`** con las siguientes propiedades:

- Puede operar como una función de un único parámetro, es decir, se puede evaluar.
- El tipo de dato del parámetro y el tipo de dato devuelto no son conocidos a priori, sino genéricos y posiblemente distintos.
- Se puede operar con él como con cualquier dato: almacenarlo, pasarlo como parámetro o devolverlo como resultado de una función.

C++ operator()

Recuerda que en C++ puedes redefinir el operador de llamada a función para cualquier objeto:

```
class plus1 {
   int operator() (int val) { return val+1; }
};

// Variable
plus1 inc;
cout << inc(3) << endl;
// In-line
cout << plus1()(4) << endl;
```

Recuerda también que en Java eso no es posible.

Problema - Functor

Define varios functores de parámetro real (`double`), que realicen las siguientes operaciones:

- Elevar al cuadrado, con resultado real:
  `sqr()`
- Multiplicar por un real fijo elegido a priori, con resultado real:
  `times(double k)`
- Redondear al entero más cercano, con resultado entero:
  `round()`

Prueba todo esto con una función `plot` (método estático en Java), que evalúe un functor en una serie de valores fijos mostrando el resultado por pantalla.

Problema - Functor

En C++ puedes encapsular el código en un espacio de nombres de esta forma:

```
namespace fn {
  template<.....>
  class functor { ... };

  class sqr   : ... { ... };
  class times : ... { ... };
  class round : ... { ... };

  void plot(.....) { .... }
}
```

Problema - Functor

```
  cout << "sqr()" << endl;
  fn::plot(fn::sqr());
sqr()
   x =  0.00  0.60  1.20  1.80  2.40  3.00  3.60  4.20  4.80  5.40  6.00
f(x) =  0.00  0.36  1.44  3.24  5.76  9.00 12.96 17.64 23.04 29.16 36.00

  cout << "times(3)" << endl;
  fn::plot(fn::times(3));
times(3)
   x =  0.00  0.60  1.20  1.80  2.40  3.00  3.60  4.20  4.80  5.40  6.00
f(x) =  0.00  1.80  3.60  5.40  7.20  9.00 10.80 12.60 14.40 16.20 18.00

  cout << "round()" << endl;
  fn::plot(fn::round());
round()
   x =  0.00  0.60  1.20  1.80  2.40  3.00  3.60  4.20  4.80  5.40  6.00
f(x) =     0     1     1     2     2     3     4     4     5     5     6
```

Problema - Predicado

Problema - Predicado

Un **predicado** es un functor que opera sobre cualquier tipo de dato, pero cuyo resultado siempre es booleano, y que evalúa alguna condición sobre su argumento. Define el tipo de datos **`predicate`**, y dos predicados sobre números enteros (`int`):

- Comprobar si un número es par: `fn::is_even()`
- Comprobar si un número es primo: `fn:is_prime()`

Problema - Predicado

Prueba estos predicados con una función `select` que muestre por pantalla aquellos números enteros entre un mínimo y un máximo que cumplan con una condición dada:

```
1 fn::select(fn::is_even(),1,10) => 2 4 6 8 10
2
3 fn::select(fn::is_prime(),1,30) => 3 5 7 11 13 17 19 23 29
```

Problema - Listas

Problema - Listas

Rediseña los algoritmos anteriores para que operen sobre listas de cualquier tipo de dato (genéricas). Define dos funciones:

- `filter`: dado un predicado y una lista, devuelve otra lista del mismo tipo de datos con los elementos que cumplen con el predicado.
- `map`: dado un functor y una lista, devuelve otra lista con los resultados de aplicar la función a cada elemento de la lista parámetro.

La verificación de la adecuación de los tipos de datos deberá realizarse en tiempo de compilación.

Problema - Listas

Ejemplo en C++:

```
1  std::list<int>  l {1,2,3,4,5,6,7,8,9,10};
2
3  std::list<int>  lfe = fn::filter(fn::is_even(),l);
4  //lfe contiene {2,4,6,8,10}
5
6  std::list<int>  lfp = fn::filter(fn::is_prime(),l);
7  //lfp contiene {3,5,7}
8
9  std::list<bool> lme = fn::map(fn::is_even(),l);
10 //lme contiene {false,true,false,true,false,true,false,true,false,true}
```

Problema - Colecciones

Rediseña `filter` para que trabaje con cualquier colección iterable además de listas.
El tipo del contenedor de salida es exactamente el mismo que el de entrada.
Ejemplo en C++:

```
1 list<int>   l {1,2,3,4,5,6,7,8,9,10};
2 auto lfe = fn::filter(fn::is_even(),l); // list<int>
3
4 vector<int> v {1,2,3,4,5,6,7,8,9,10};
5 auto vfe = fn::filter(fn::is_even(),v); // vector<int>
```

Problema - Colecciones

Intenta hacer los mismo con `map`: el tipo de contenedor de salida debe ser el mismo que el de entrada, pero con otro tipo de contenido.

¿ Que problemas encuentras ?

Pista (C++ ++ Advanced Plus...): *template template parameters*...

Problema - Trabajo futuro

Problema - Trabajo futuro

Por si tienes (mucho) tiempo libre...

- Si tus soluciones se basan en programación genérica, intenta rehacerlas utilizando herencia (o viceversa).
- Rehaz tus soluciones en otro lenguaje orientado a objetos (para explorar las diferencias entre lenguajes).
- Comprueba otros algoritmos similares que ya estén en la biblioteca estándar de los lenguajes (como `std::function` o `std::transform`).
  ¿Qué diferencias y similitudes encuentas?

Functores

Problemas

Juan Magallón – Adolfo Muñoz

Grado en Ingeniería Informática

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza