Лекция 10-11. Элементы функционального программирования

Вывод типов (Type inference)

def: автоматическое полное или частичное вычисление типов некоторых выражений программы, основанное лишь на ее синтаксической структуре и производимое статически, без необходимости запуска программы.

примеры:

- dot(v1,v2) = v1[0]*v2[0] + v1[1]*v2[1]
- `order.getCustomer().getName()` тут из типа `order` автоматически выводится тип выражения `order.getCustomer()` и компилятор убеждается, что для этого типа определена операция `getName()`
- C++ может выводить типы шаблонных аргументов шаблонных функций из типов их фактических параметров («снизу вверх»).

Функции высших порядков

функция: отображение A->B

def: Функция называется функцией высшего порядка, если один из её аргументов — это функция, либо её возвращаемое значение — это функция.

пример 1: оператор сортировки, фильтр в C++

пример 2: `ORDER BY`

пример 3:

- функциональный объект в C++

Пример 4:

- pthreads

Замыкания (Closure)

def: замыкание — это значение (обычно процедура), имеющее доступ к переменным из области видимости, в которой оно было создано.

Википедия:

- Замыкание, так же как и экземпляр объекта, есть способ представления функциональности и данных, связанных и упакованных вместе
- Замыкание — это особый вид функции. Она определена в теле другой функции и создаётся каждый раз во время её выполнения. Синтстически это выглядит как функция, находящаяся целиком в теле другой функции. При этом вложенная внутренняя функция содержит ссылки на локальные переменные внешней функции. Каждый раз при выполнении внешней функции происходит создание нового экземпляра внутренней функции, с новыми ссылками на переменные внешней функции.
- В случае замыкания ссылки на переменные внешней функции действительны внутри вложенной функции до тех пор, пока работает вложенная функция, даже если внешняя функция закончила работу, и переменные вышли из области видимости.
- Девид Мертц приводит следующее определение замыкания: "Замыкание - это процедура вместе с привязанной к ней совокупностью данных". Замыкание связывает код функции с её лексическим окружением (местом, в котором она определена в коде). Лексические переменные замыкания отличаются от глобальных переменных тем, что они не занимают глобальное пространство имён. От переменных в объектах они отличаются тем, что привязаны к функциям, а не объектам.

(в противовес объектам в объектном программировании, как: "данные вместе с привязанным к ним совокупностью процедур").

- пример:

```
class CalculationWindow extends JFrame {
    private JButton btnSave;
    ...

    public final void calculateInSeparateThread(final URI uri) {
        // Выражение "new Thread() { ... }" представляет собой пример анонимного класса.
        new Thread() {
            public void run() {
                // Имеет доступ к финальным (final) переменным:
                calculate(uri);
                // Имеет доступ к приватным членам содержащего класса:
                btnSave.setEnabled(true);
            }
        }.start();
    }
}
```

еще пример на Python

```
def multiplier(n):
    def mul(x):
      return x*n;
    return mul


  mul3 = multiplier(3)
  mul5 = multiplier(5)

  print(mul3(3),mul3(5))
  print(mul5(3),mul5(5))
  print(mul3(3),mul3(5))
```

генерация

```
def makehello(prefix):
 def hello(name):
  print(prefix+' '+name)
 return hello;
privet = makehello("Privet")
hello  = makehello("Hello");
privet("aa");
hello("aa");
```

Разное

Upward funarg problem (наружняя фунарг-проблема)[5]: как реализовать возврат замыкания в качестве результата? Поскольку замыкание использует переменные, связанные в области видимости, где оно было создано (или еще выше), то цикл жизни этих переменных должен продлиться, по крайней мере, до последнего использования этого

замыкания — возможно, даже после выхода из создавшей их области видимости (например, после возврата из процедуры, возвратившей это замыкание).

Downward funarg problem (внутренняя фунарг-проблема): как реализовать возможность передачи замыкания в качестве аргумента процедуре? Проблема в данном случае состоит лишь в том, как обеспечить одинаковое представление обычных процедур и замыканий, так, чтобы процедура могла принимать в качестве аргумента другую процедуру, не заботясь о том, является ли она замыканием. В отличие от upward funarg problem, в данном случае предполагается, что передаваемое замыкание не сохраняется процедурой в глобальных переменных, и, таким образом, оно существует лишь во время ее вызова: благодаря этому исчезают проблемы управления памятью.

Вопросы для проверки:

- Может ли существовать замыкание, после после того как функция в которой оно было создано завершит работу.

Источники

- http://fprog.ru/2009/issue3/eugene-kirpichov-elements-of-functional-languages/