Лабораторная работа № 5

Исследование особенностей реализации операций каррирование, частичного применения функций и замыканий в языке Scala

1. Цель работы

Исследовать понятия каррирования, частичного применения и замыкания в языке Scala и получить практические навыки их применения.

2. Общие положения

2.1. Частичное применение

В функциональном программировании при вызове функции можно перечислить не все аргументы, а только их часть, т. е. несколько первых аргументов мы можем указать, а остальные отбросить. Таким образом, результатом вызова такой функции будет являться функция, в которую необходимо передать остальные аргументы. Такая функция, результат вызова которой будет представлять собой функцию, в которую можно передать все оставшиеся аргументы называется частично применённой.

2.1.1 Частичное применение функции в Scala

Язык Scala также допускает частичное применение функций. Рассмотрим пример, представленный ниже:

```
def price(product : String) : Double =
  product match {
    case "apples" => 140
    case "oranges" => 223
}

def withTax(cost: Double, state: String) : Double =
  state match {
    case "NY" => cost * 2
    case "FL" => cost * 3
}

val locallyTaxed = withTax(_: Double, "NY")
val costOfApples = locallyTaxed(price("apples"))

// выдаст ошибку java.lang.AssertionError: assertion failed
// если условие не выполнится
assert(Math.round(costOfApples) == 280)
```

В примере сначала создаётся функция price, которая возвращает отображение между product (товар) и price (цена). Затем объявляется функция withTax(), которая принимает аргументы cost и state. Пусть для определенного

исходного файла известно, что вычисления будут производиться с налогами (taxes) только одного штата (state). Вместо того чтобы "каррировать" лишний аргумент при каждом вызове, можно "частично применить" аргумент state и возвратить версию функции, в которой значение state зафиксировано. Функция locallyTaxed принимает единственный аргумент cost.

2.2. Каррирование

2.2.1. Каррирование в λ-исчислении

Если функция f имеет тип A1 → (A2 → (. . .(An → B). . .)), то, чтобы полностью вычислить значение f(a1, a2, . . . , an), необходимо последовательно провести вычисление (. . .(f(a1)a2). . .)an. И результатом вычисления будет объект типа B. Соответственно, выражение, в котором все функции рассматриваются как функции одного аргумента, а единственной операцией является аппликация (применение), называются выражениями в форме «оператор — операнд». Такие функции получили название «каррированные», а сам процесс сведения типа функции к виду, приведенному выше — каррирование в честь Хаслелля Карри, который переоткрыл, развил и популяризовал данное понятие (хотя первоотрывателем является Моисей Эльевич Шейнфинкель).

Если вспомнить λ-исчисление то обнаружится, что в нем уже есть математическая абстракция для аппликативных форм записей. Например:
$f(x) = x^2 + 5 \Rightarrow \lambda x.(x^2 + 5)$,
$f(x, y) = x + y \Rightarrow \lambda y.\lambda x.(x + y)$,
$f(x, y, z) = x^2 + y^2 + z^2 \Rightarrow \lambda z.\lambda y.\lambda x.(x^2 + y^2 + z^2)$.

Таким образом, каррирование по сути образует цепочку функций с одним аргументом.

2.2.2. Каррирование в Scala

В Scala функции позволяют задавать несколько списков аргументов виде наборов круглых скобок. Если вы вызываете какую-либо функцию с меньшим количеством аргументов, чем задано для нее, эта функция берет списки опущенных аргументов в качестве своих аргументов. Рассмотрим пример из документации Scala:

```
def filter(xs: List[Int], p: Int => Boolean): List[Int] =
  if (xs.isEmpty) xs
  else if (p(xs.head)) xs.head :: filter(xs.tail, p)
  else filter(xs.tail, p)

def modN(n: Int)(x: Int) = ((x % n) == 0)

val nums = List(1, 2, 3, 4, 5, 6, 7, 8)
println(filter(nums, modN(2)))
println(filter(nums, modN(3)))
```

В примере функция filter() рекурсивно применяет переданные критерии фильтрации. Функция modN() определена с двумя списками аргументов. При вызове функции modN с использованием функции filter(), передаётся единственный аргумент. Функция filter() принимает в качестве своего второго аргумента функцию с аргументом Int и возвращает значение Boolean соответствующее сигнатуре каррированной функции, которая и передаётся. Таким образом,функция modN берёт недостающие аргументы уже будучи вызванной внутри функции filter, но уже под другим именем — «p».

2.3. Замыкания

2.3.1. Замыкания в λ-выражениях

Замыкания это некая функция с какими-то переменными, но эти переменные не являются локальными для данной функции, а определены извне, т. е. в той же области видимости, где определена функция. Такие переменные принято называть контекстом.

Замыкание — (англ closures) это лямбда-выражение, которое содержит помимо связанной переменной и тела функции еще и контекст.

Замыкание тесно связано с понятием слабой заголовочной нормальной формы.

Выражение, не имеющее свободных переменных (замкнутое) находится в слабой заголовочной нормальной форме (СЗНФ), если оно имеет один из следующих видов:

- константа: c;
- определение функции: λx.E;
- частичное применение функции: P E_1 E_2 … E_k

Например, выражение λx.((λy. λx.+ x y) x) находится в СЗНФ.

Вычисления, происходящие при исполнении программы в «ленивых» вычислениях, соответствуют редукции выражения в нормальном порядке до приведения к СЗНФ, дополненной эффектом «разделения» значений переменных при подстановке аргумента, еще не находящегося в СЗНФ.

2.3.2. Замыкания в Scala

Пример простого замыкания представлен ниже:

```
val outer = 10
val myFuncLiteral = (y: Int) => y * outer
val result = myFuncLiteral(2)
```

После выполнения данного фрагмента кода функция myFuncLiteral «захватит» из области видимости своего определения значение переменной outer и в результате result будет равен 20.

3. Порядок выполнения работы

3.1. Реализовать на языке Scala функцию, возвращающую количество собственных вызовов.

3.2. Реализовать функцию на Scala, используя механизм каррирования:

Вариант 1: Функция возведения в степень. Пример вызова функции на псевдокоде:

возведениеВСтепень(3)(2)(1)(0) → 1.

Вариант 2: Функция перевода десятичного числа в n-ичное представление. Пример вызова:

десятичноеЧислоВNичное(число, порядок_числа).

Используя каррирование, напишите функцию преобразования десятичного числа в двоичное.

3.3. Реализовать функцию, которая принимает один параметр - начальное число Z - и возвращает функцию, которая принимает один параметр - длину списка - и возвращает список заданной длины, содержащий случайные числа. Модуль разности случайных чисел и начального числа Z не должен превышать 5.

3.4. Составить отчет, содержащий результаты выполнения программы.

4. Содержание отчета

4.1. Цель работы.
4.2. Задание на работу.
4.3. Исходный код программы.
4.4. Результаты работы программы.
4.5. Выводы по работе.

5. Контрольные вопросы

5.1. Что такое каррирование?
5.2. Объясните механизм частичного применения в Scala.
5.3. Чем каррирование отличается от частичного применения?
5.4. Что такое замыкание? Приведите пример замыкания.