# Функциональные множества

В данной лабораторной работе мы будем иметь дело с множествами целых чисел. Однако, в отличие от «стандартной» реализации, которую изучают в курсе «Структуры данных», мы не будем хранить числа, относящиеся к множеству, напрямую. Вместо этого мы опишем множество через его характеристическую функцию, то есть функцию, которая при передаче ей в качестве параметра целого числа, вернет нам логический признак: принадлежит число множеству или нет.

Например, для множества всех четных чисел такая функция будет выглядеть как

(x: Int) => x % 2 == 0

В языке Scala, мы представим такое множество через функциональный тип «*Set*», заданный как

type Set = Int => Boolean

и, используя это описание, построим функцию «*contains*», проверяющую принадлежность целого числа заданному характеристической функцией множеству

def contains(s: Set, elem: Int): Boolean = s(elem)

## Задание к лабораторной работе

Реализуйте описанные ниже функции, не сохраняя значения элементов множества в структурах данных, а используя только характеристические функции, их вызовы и операции над ними (над функциями).

При решении задачи имейте в виду, что большинство требуемых функций могут быть реализованы в одну строку, так что прежде чем писать много кода, – хорошо подумайте.

### Часть №1. Базовый набор функций

Определим основные функции, позволяющие создавать и оперировать множествами.

Для начала опишем функцию «*singletonSet*» создающую множество, состоящее из единственного целого числа. Она позволит нам конструировать примитивные множества, из которых мы будем путем комбинирования строить более сложные наборы.

def singletonSet(elem: Int): Set

Следующим шагом опишем три стандартные операции над множествами: объединение (функция «*union*»), пересечение (функция «*intersect*») и разность (функция «*diff*»). Каждая из них получает на вход два множества (также заданные характеристическими функциями) и возвращает новое множество (новую характеристическую функцию) соответствующее проводимой операции.

def union(s: Set, t: Set): Set

def intersect(s: Set, t: Set): Set

def diff(s: Set, t: Set): Set

Наконец, создадим функцию «*filter*», которая выбирает из заданного множества только те элементы, которые удовлетворяют условному предикату. Результатом фильтрации должно быть новое множество.

def filter(s: Set, p: Int => Boolean): Set

### Часть №2. Запросы и трансформация множеств

В данной части задания мы сосредоточимся на функциях позволяющих выполнять запросы или изменения элементов множества. Ключевой будет функция «*forall*» проверяющая удовлетворяют ли все элементы заданного множества некоторому условному предикату.

def forall(s: Set, p: Int => Boolean): Boolean

К сожалению, наше множество не позволяет напрямую перебирать элементы, содержащиеся в нем. Единственное, что мы можем сделать с множеством – это проверить, принадлежит ли ему некоторое целое число. Так что если мы хотим выполнить некоторые действия над всеми элементами множества, мы должны сначала перебрать все целые числа и выполнить действия только над теми, которые входят во множество. Такой перебор (даже в рамках типа Int) займет слишком много времени, и для сокращения перебора в данной лабораторной работе мы будем считать, что все целые числа ограничены интервалом от -1000 до 1000 включительно (*-1000 <= x <= 1000*).

Итак, в качестве первого задания завершите код функции «*forall*» в файле «FunSets.scala».

Следующим шагом, используя функцию «*forall*», реализуйте функцию «*exists*», проверяющую наличие во множестве хотя бы одного элемента, удовлетворяющего условному предикату (подсказка: помочь в реализации могут знания из области математической логики).

def exists(s: Set, p: Int => Boolean): Boolean

В качестве последней, напишите функцию «*map*» преобразующую исходное множество путем применения к каждому его элементу некоторой трансформирующей функции (параметр «*f*»).

def map(s: Set, f: Int => Int): Set