МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики

ОТЧЁТ

по лабораторной работе № 4

«Алгебраические типы данных и классы типов»

по дисциплине «Функциональное программирование»

Выполнил: студент группы ПМИМ-61

Резанко А. А.

Проверил: _____

Новосибирск, 2026

Лабораторная работа № 4

Алгебраические типы данных и классы типов

Цель работы

Научиться определять собственные типы данных конструкцией data – перечисления, записи, суммы и произведения типов, – объявлять экземпляры классов типов вручную и получать их автоматически.

Теоретические сведения

Объявление data вводит новый тип и перечисляет его конструкторы. Конструктор без аргументов задаёт одно значение, конструктор с аргументами – произведение типов: значение хранит все поля сразу. Несколько конструкторов образуют сумму типов: значение соответствует ровно одной альтернативе, а какой именно – выясняется сопоставлением с образцом.

Класс типов описывает интерфейс: набор сигнатур, которые тип обязан реализовать, чтобы стать его экземпляром. Один класс отвечает за сравнение на равенство, другой за упорядоченность, третий за строковое представление, ещё два за перечислимость и наличие границ. Конструкция deriving поручает генерацию экземпляра компилятору; при нестандартной семантике экземпляр пишется вручную.

Средства и порядок выполнения

Компилятор GHC 9.4.7, операционная система Ubuntu 24.04. Исходный текст программы размещён в файле Lab4.hs и приведён полностью в приложении А. Сборка и запуск:

```
ghc -O0 -Wall Lab4.hs -o lab4
./lab4
```

Ключ -Wall включает полный набор предупреждений компилятора; сборка проходит без замечаний. В программу встроено 20 проверок: каждая сверяет результат функции с ожидаемым значением и печатает отметку в

протоколе. Вместо экранных снимков в отчёте приведён текстовый протокол консольного вывода – он воспроизводится запуском программы и содержит те же данные.

Ход работы

4.1 Перечисления

Тип дня недели объявлен перечислением из семи конструкторов без аргументов. Список в deriving определяет, что компилятор сгенерирует сам: печать имени конструктора, сравнение значений в порядке объявления, нумерацию с диапазонами и границы.

Проверка выходного дня разбирает аргумент сопоставлением с образцом: два конкретных конструктора дают истину, шаблон подчёркивания закрывает оставшиеся пять случаев.

Функция перехода к следующему элементу на последнем конструкторе завершается исключением. Чтобы неделя замыкалась и за воскресеньем следовал понедельник, переход сравнивает аргумент с верхней границей – ради этого в список deriving добавлен класс границ.

Нумерация конструкторов начинается с нуля, поэтому порядковый номер дня получается прибавлением единицы: понедельник получает номер 1, воскресенье – 7.

Листинг:

```
data Day = Monday | Tuesday | Wednesday | Thursday
         | Friday | Saturday | Sunday
  deriving (Show, Eq, Ord, Enum, Bounded)

-- Выходной день
isWeekend :: Day -> Bool
isWeekend Saturday = True
isWeekend Sunday   = True
isWeekend _        = False

-- Следующий день. succ на maxBound бросает исключение,
-- поэтому неделя замыкается вручную: за воскресеньем идёт понедельник.
nextDay :: Day -> Day
nextDay d
  | d == maxBound = minBound
  | otherwise     = succ d

-- Порядковый номер дня недели, 1..7 (fromEnum нумерует с нуля)
dayNumber :: Day -> Int
dayNumber d = fromEnum d + 1
```

Результат работы:

```
Monday     номер 1  выходной: False  следующий: Tuesday
Tuesday    номер 2  выходной: False  следующий: Wednesday
Wednesday  номер 3  выходной: False  следующий: Thursday
Thursday   номер 4  выходной: False  следующий: Friday
Friday     номер 5  выходной: False  следующий: Saturday
Saturday   номер 6  выходной: True  следующий: Sunday
Sunday     номер 7  выходной: True  следующий: Monday
Сравнение из Ord:   Monday < Friday = True
Диапазон из Enum:   [Monday .. Wednesday] = [Monday,Tuesday,Wednesday]
```

4.2 Записи

Тип студента объявлен в синтаксисе записей. Имена полей служат одновременно функциями-селекторами: обращение по имени поля извлекает значение из записи.

В среднем балле оценки приводятся к вещественному типу: деление требует класса дробных чисел, тогда как сумма и длина списка целочисленные. Пустой список оценок дал бы неопределённость, поэтому он отсечён охраняющим условием.

Выбор лучшего студента сделан стандартной функцией максимума по компаратору. Для пустого списка результат не определён: вместо невнятного сообщения из недр библиотеки вызывается ошибка с пояснением.

Замена оценки возвращает новую запись синтаксисом обновления полей. Значения в Haskell неизменяемы, поэтому исходная запись остаётся прежней – это подтверждает отдельная проверка. Список разрезается по индексу, старая оценка отбрасывается; индекс вне диапазона оставляет запись без изменений.

Листинг:

```
data Student = Student
  { studentName   :: String
  , studentGroup  :: String
  , studentGrades :: [Int]
  } deriving (Show)

alice, bob, carol :: Student
alice = Student "Алиса" "ПМИ-1" [5, 4, 5, 5, 3]
bob   = Student "Боб"   "ПМИ-1" [4, 4, 3, 5, 4]
carol = Student "Кэрол" "ПМИ-2" [5, 5, 5, 5, 5]

students :: [Student]
students = [alice, bob, carol]

-- Средний балл; у студента без оценок считаем 0, чтобы не получить NaN
averageGrade :: Student -> Double
averageGrade s
  | null gs   = 0
```

```
    | otherwise = fromIntegral (sum gs) / fromIntegral (length gs)
    where gs = studentGrades s

-- Студент с наивысшим средним баллом
topStudent :: [Student] -> Student
topStudent [] = error "topStudent: список студентов пуст"
topStudent ss = maximumBy (comparing averageGrade) ss

-- Заменить оценку по индексу (индексация с нуля).
-- Индекс вне диапазона оставляет запись без изменений.
updateGrade :: Student -> Int -> Int -> Student
updateGrade s i g
  | i < 0 || i >= length gs = s
  | otherwise               = s { studentGrades = before ++ g : drop 1 rest }
  where
    gs             = studentGrades s
    (before, rest) = splitAt i gs
```

Результат работы:

```
Алиса  ПМИ-1  [5,4,5,5,3]       среднее 4.40
Боб    ПМИ-1  [4,4,3,5,4]       среднее 4.00
Кэрол  ПМИ-2  [5,5,5,5,5]       среднее 5.00
Лучший студент: Кэрол (5.00)
updateGrade alice 4 5: [5,4,5,5,3] -> [5,4,5,5,5]
индекс вне диапазона:  [5,4,5,5,3]
```

4.3 Сумма типов

Тип фигуры – сумма трёх альтернатив, каждая из которых является произведением: круг хранит одно число, прямоугольник два, треугольник три. Площадь и периметр разбирают значение по конструктору, и полнота набора образцов контролируется компилятором: пропущенная ветка вызвала бы предупреждение при ключе -Wall.

Площадь треугольника вычисляется по формуле Герона через полупериметр. Показатель степени в площади круга записан с явной аннотацией типа, иначе -Wall сообщает о неоднозначности типа числового литерала.

Листинг этого задания в методических указаниях не компилируется: в заголовке образца для треугольника пропущена закрывающая скобка. В работе ошибка исправлена.

Формула Герона на тройке чисел, нарушающей неравенство треугольника, даёт корень из отрицательного числа. Для контроля добавлена проверка корректности фигуры до вычислений.

Листинг:

```
data Shape
  = Circle Double              -- радиус
  | Rectangle Double Double    -- ширина, высота
  | Triangle Double Double Double  -- стороны a, b, c
```

```
  deriving (Show)

area :: Shape -> Double
area (Circle r)       = pi * r ^ (2 :: Int)
area (Rectangle w h)  = w * h
area (Triangle a b c) = sqrt (p * (p - a) * (p - b) * (p - c))
  where p = (a + b + c) / 2       -- полупериметр, формула Герона

perimeter :: Shape -> Double
perimeter (Circle r)       = 2 * pi * r
perimeter (Rectangle w h)  = 2 * (w + h)
perimeter (Triangle a b c) = a + b + c

-- Неравенство треугольника: для вырожденных троек площадь по Герону даёт NaN
isValidShape :: Shape -> Bool
isValidShape (Circle r)       = r > 0
isValidShape (Rectangle w h)  = w > 0 && h > 0
isValidShape (Triangle a b c) =
  a > 0 && b > 0 && c > 0 && a + b > c && a + c > b && b + c > a
```

Результат работы:

```
  Circle 2.0             площадь  12.5664  периметр  12.5664
  Rectangle 3.0 4.0      площадь  12.0000  периметр  14.0000
  Triangle 3.0 4.0 5.0   площадь   6.0000  периметр  12.0000
  Triangle 1 2 10 корректен: False
```

4.4 Ручной экземпляр класса равенства

Экземпляр класса равенства для типа цвета объявлен вручную. Достаточно определить одну операцию: в самом классе равенство и неравенство заданы друг через друга определениями по умолчанию, поэтому неравенство получено без дополнительного кода.

Тело экземпляра сравнивает три компоненты цвета покомпонентно – ровно то, что сгенерировал бы deriving. Ручное объявление оправдано, когда требуется иная семантика: например, сравнение цветов с допуском или с точностью до нормировки.

Листинг:

```
data Color = RGB Int Int Int
  deriving (Show)

instance Eq Color where
  RGB r1 g1 b1 == RGB r2 g2 b2 =
    r1 == r2 && g1 == g2 && b1 == b2
```

Результат работы:

```
  RGB 255 0 0 == RGB 255 0 0 -> True
  RGB 255 0 0 == RGB 0 0 255 -> False
  /= получен бесплатно из определения класса: True
```

4.5 Рекурсивные типы данных

Тип дерева рекурсивен: конструктор узла содержит два значения того же типа. Значения хранятся только в листьях, внутренние узлы задают структуру.

Размер считает листья, глубина возвращает число уровней, причём одиночный лист считается деревом глубины 1. Для контрольного дерева из четырёх листьев получаются 4 и 3 соответственно.

Обход собирает значения слева направо. Конкатенация списков имеет линейную стоимость от длины левого аргумента, поэтому на сбалансированном дереве обход обходится дороже линейного; версия с накопителем убирает лишний множитель, но здесь важнее прозрачность записи.

Отображение сохраняет форму дерева и заменяет только значения в листьях. Сигнатура допускает смену типа: вызов с функцией печати превращает дерево чисел в дерево строк, что и показано в протоколе.

Листинг:

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
  deriving (Show)

-- Количество листьев
treeSize :: Tree a -> Int
treeSize (Leaf _)   = 1
treeSize (Node l r) = treeSize l + treeSize r

-- Глубина: одиночный лист считаем деревом глубины 1
treeDepth :: Tree a -> Int
treeDepth (Leaf _)   = 1
treeDepth (Node l r) = 1 + max (treeDepth l) (treeDepth r)

-- Обход in-order: значения хранятся только в листьях,
-- поэтому обход сводится к «сначала левое поддерево, затем правое»
flattenTree :: Tree a -> [a]
flattenTree (Leaf x)   = [x]
flattenTree (Node l r) = flattenTree l ++ flattenTree r

mapTree :: (a -> b) -> Tree a -> Tree b
mapTree f (Leaf x)   = Leaf (f x)
mapTree f (Node l r) = Node (mapTree f l) (mapTree f r)

sampleTree :: Tree Int
sampleTree = Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4))
```

Результат работы:

```
  sampleTree  = Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4))
  treeSize    = 4
  treeDepth   = 3
  flattenTree = [1,2,3,4]
  mapTree (*10) = [10,20,30,40]
```

```
mapTree show  = ["1","2","3","4"]
```

Тестирование

Полный вывод программы за один запуск, включая встроенные проверки:

```
=== 4.1. Перечисления: Day ===
  Monday     номер 1  выходной: False  следующий: Tuesday
  Tuesday    номер 2  выходной: False  следующий: Wednesday
  Wednesday  номер 3  выходной: False  следующий: Thursday
  Thursday   номер 4  выходной: False  следующий: Friday
  Friday     номер 5  выходной: False  следующий: Saturday
  Saturday   номер 6  выходной: True  следующий: Sunday
  Sunday     номер 7  выходной: True  следующий: Monday
  Сравнение из Ord:   Monday < Friday = True
  Диапазон из Enum:   [Monday .. Wednesday] = [Monday,Tuesday,Wednesday]

=== 4.2. Записи: Student ===
  Алиса  ПМИ-1  [5,4,5,5,3]       среднее 4.40
  Боб    ПМИ-1  [4,4,3,5,4]       среднее 4.00
  Кэрол  ПМИ-2  [5,5,5,5,5]       среднее 5.00
  Лучший студент: Кэрол (5.00)
  updateGrade alice 4 5: [5,4,5,5,3] -> [5,4,5,5,5]
  индекс вне диапазона:  [5,4,5,5,3]

=== 4.3. Сумма типов: Shape ===
  Circle 2.0             площадь  12.5664  периметр  12.5664
  Rectangle 3.0 4.0      площадь  12.0000  периметр  14.0000
  Triangle 3.0 4.0 5.0   площадь   6.0000  периметр  12.0000
  Triangle 1 2 10 корректен: False

=== 4.4. Ручной экземпляр Eq: Color ===
  RGB 255 0 0 == RGB 255 0 0 -> True
  RGB 255 0 0 == RGB 0 0 255 -> False
  /= получен бесплатно из определения класса: True

=== 4.5. Рекурсивные типы: Tree ===
  sampleTree  = Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4))
  treeSize    = 4
  treeDepth   = 3
  flattenTree = [1,2,3,4]
  mapTree (*10) = [10,20,30,40]
  mapTree show  = ["1","2","3","4"]

=== Самопроверка ===
  [OK  ] isWeekend Saturday
  [OK  ] not (isWeekend Monday)
  [OK  ] nextDay Sunday == Monday
  [OK  ] nextDay Monday == Tuesday
  [OK  ] dayNumber Sunday == 7
  [OK  ] averageGrade alice == 4.4
  [OK  ] topStudent == Кэрол
  [OK  ] updateGrade заменяет оценку
  [OK  ] updateGrade не портит исходную запись
  [OK  ] area (Circle 2) == 4*pi
  [OK  ] area (Rectangle 3 4) == 12
  [OK  ] area (Triangle 3 4 5) == 6
  [OK  ] perimeter (Triangle 3 4 5) == 12
  [OK  ] RGB 255 0 0 == RGB 255 0 0
  [OK  ] RGB 255 0 0 /= RGB 0 0 255
  [OK  ] treeSize sampleTree == 4
  [OK  ] treeDepth sampleTree == 3
  [OK  ] flattenTree sampleTree == [1,2,3,4]
  [OK  ] mapTree сохраняет форму
  [OK  ] mapTree (*10) даёт [10,20,30,40]

Пройдено 20 из 20 проверок.
```

Все 20 проверок пройдены (20 из 20), расхождений с ожидаемыми значениями нет.

Ответы на контрольные вопросы

1. Чем отличается data от type и newtype?

Объявление type вводит синоним: новое имя для уже существующего типа. Синоним взаимозаменяем с исходным типом, собственных конструкторов не имеет и не может получить отдельный экземпляр класса.

Объявление data создаёт новый тип с произвольным числом конструкторов и полей. Значение такого типа представлено в памяти отдельным объектом с тегом конструктора, на него можно объявлять любые экземпляры классов.

Объявление newtype – частный случай data ровно с одним конструктором и одним полем. Компилятор стирает обёртку при трансляции, поэтому обращение к значению не стоит ничего, а тип остаётся отдельным и допускает собственные экземпляры. Отличие проявляется и в строгости: сопоставление с образцом по конструктору newtype не вычисляет аргумент, тогда как для data вычисление тега обязательно.

2. Что выводит deriving для класса равенства и как он это делает?

Компилятор порождает объявление экземпляра со структурным сравнением: сначала сверяются теги конструкторов, при их совпадении поля сравниваются попарно слева направо операцией равенства соответствующих типов. Значения разных конструкторов считаются неравными.

Для этого все типы полей сами обязаны быть экземплярами этого класса. У параметрических типов компилятор добавляет контекст: для дерева из задания 4.5 сгенерированный экземпляр требовал бы равенства для типа значений в листьях. Неравенство отдельно не генерируется – оно берётся из определения по умолчанию в самом классе.

3. Разница между суммой и произведением типов. Какая структура у типа фигуры? А у типа студента?

Произведение означает, что значение содержит все перечисленные поля одновременно, и число возможных значений равно произведению мощностей типов полей. Сумма означает выбор одной альтернативы из нескольких, и число значений равно сумме мощностей вариантов.

Тип студента – чистое произведение: один конструктор с тремя полями, то есть строка, строка и список целых одновременно. Тип фигуры – сумма трёх альтернатив, каждая из которых внутри себя является произведением: одно вещественное число, пара чисел или тройка чисел. Обработка произведения сводится к обращению к полям, обработка суммы требует разбора по всем конструкторам.

4. Является ли дерево рекурсивным типом? Почему Haskell позволяет это?

Да, тип рекурсивен: в объявлении конструктора узла участвует сам объявляемый тип. Имя типа становится доступным уже внутри собственного объявления, и никаких ограничений на его использование там нет.

Допустимость такого объявления обеспечивается представлением данных: поля конструктора хранятся как ссылки на объекты в куче, поэтому размер значения не зависит от глубины дерева и известен компилятору заранее. Ленивая семантика вычислений идёт дальше и допускает бесконечные рекурсивные структуры: они разворачиваются ровно настолько, насколько их затребовал потребитель.

Выводы

В работе освоено объявление алгебраических типов всех трёх видов: перечисления, записи и суммы типов с рекурсивным деревом. Показано, что выбор между deriving и ручным объявлением экземпляра определяется семантикой: структурное сравнение компилятор генерирует сам, нестандартное приходится описывать явно.

Сопоставление с образцом оказывается основным средством разбора суммы типов, а контроль полноты образцов компилятором переводит часть ошибок из времени выполнения во время сборки. Неизменяемость значений заставляет выражать изменение как построение новой записи, что подтверждено проверкой сохранности исходных данных после замены оценки.

Программа собирается без предупреждений при ключе -Wall и проходит все 20 встроенных проверок.

ПРИЛОЖЕНИЕ А. Полный листинг программы (Lab4.hs)

```
-- Лабораторная работа №4. Алгебраические типы данных и классы типов
-- Резанко А. А., ПМИ-23
--
-- Сборка:  ghc -O0 -Wall Lab4.hs -o lab4
-- Запуск:  ./lab4

module Main where

import Data.List (maximumBy)
import Data.Ord (comparing)
import System.IO (hSetEncoding, stdout, utf8)
import Text.Printf (printf)

-- ==================== 4.1. Перечисления ====================

data Day = Monday | Tuesday | Wednesday | Thursday
         | Friday | Saturday | Sunday
  deriving (Show, Eq, Ord, Enum, Bounded)

-- Выходной день
isWeekend :: Day -> Bool
isWeekend Saturday = True
isWeekend Sunday   = True
isWeekend _        = False

-- Следующий день. succ на maxBound бросает исключение,
-- поэтому неделя замыкается вручную: за воскресеньем идёт понедельник.
nextDay :: Day -> Day
nextDay d
  | d == maxBound = minBound
  | otherwise     = succ d

-- Порядковый номер дня недели, 1..7 (fromEnum нумерует с нуля)
dayNumber :: Day -> Int
dayNumber d = fromEnum d + 1

-- ==================== 4.2. Записи ====================

data Student = Student
  { studentName   :: String
  , studentGroup  :: String
  , studentGrades :: [Int]
  } deriving (Show)

alice, bob, carol :: Student
alice = Student "Алиса" "ПМИ-1" [5, 4, 5, 5, 3]
bob   = Student "Боб"   "ПМИ-1" [4, 4, 3, 5, 4]
carol = Student "Кэрол" "ПМИ-2" [5, 5, 5, 5, 5]

students :: [Student]
students = [alice, bob, carol]

-- Средний балл; у студента без оценок считаем 0, чтобы не получить NaN
averageGrade :: Student -> Double
averageGrade s
  | null gs   = 0
  | otherwise = fromIntegral (sum gs) / fromIntegral (length gs)
  where gs = studentGrades s

-- Студент с наивысшим средним баллом
topStudent :: [Student] -> Student
topStudent [] = error "topStudent: список студентов пуст"
topStudent ss = maximumBy (comparing averageGrade) ss

-- Заменить оценку по индексу (индексация с нуля).
-- Индекс вне диапазона оставляет запись без изменений.
updateGrade :: Student -> Int -> Int -> Student
updateGrade s i g
```

```
  | i < 0 || i >= length gs = s
  | otherwise               = s { studentGrades = before ++ g : drop 1 rest }
  where
    gs             = studentGrades s
    (before, rest) = splitAt i gs

-- ====================== 4.3. Сумма типов ======================

data Shape
  = Circle Double              -- радиус
  | Rectangle Double Double    -- ширина, высота
  | Triangle Double Double Double  -- стороны a, b, c
  deriving (Show)

area :: Shape -> Double
area (Circle r)       = pi * r ^ (2 :: Int)
area (Rectangle w h)  = w * h
area (Triangle a b c) = sqrt (p * (p - a) * (p - b) * (p - c))
  where p = (a + b + c) / 2       -- полупериметр, формула Герона

perimeter :: Shape -> Double
perimeter (Circle r)       = 2 * pi * r
perimeter (Rectangle w h)  = 2 * (w + h)
perimeter (Triangle a b c) = a + b + c

-- Неравенство треугольника: для вырожденных троек площадь по Герону даёт NaN
isValidShape :: Shape -> Bool
isValidShape (Circle r)       = r > 0
isValidShape (Rectangle w h)  = w > 0 && h > 0
isValidShape (Triangle a b c) =
  a > 0 && b > 0 && c > 0 && a + b > c && a + c > b && b + c > a

-- ====================== 4.4. Ручной экземпляр Eq ======================

data Color = RGB Int Int Int
  deriving (Show)

instance Eq Color where
  RGB r1 g1 b1 == RGB r2 g2 b2 =
    r1 == r2 && g1 == g2 && b1 == b2

-- ====================== 4.5. Рекурсивные типы ======================

data Tree a = Leaf a | Node (Tree a) (Tree a)
  deriving (Show)

-- Количество листьев
treeSize :: Tree a -> Int
treeSize (Leaf _)   = 1
treeSize (Node l r) = treeSize l + treeSize r

-- Глубина: одиночный лист считаем деревом глубины 1
treeDepth :: Tree a -> Int
treeDepth (Leaf _)   = 1
treeDepth (Node l r) = 1 + max (treeDepth l) (treeDepth r)

-- Обход in-order: значения хранятся только в листьях,
-- поэтому обход сводится к «сначала левое поддерево, затем правое»
flattenTree :: Tree a -> [a]
flattenTree (Leaf x)   = [x]
flattenTree (Node l r) = flattenTree l ++ flattenTree r

mapTree :: (a -> b) -> Tree a -> Tree b
mapTree f (Leaf x)   = Leaf (f x)
mapTree f (Node l r) = Node (mapTree f l) (mapTree f r)

sampleTree :: Tree Int
sampleTree = Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Leaf 4))

-- ====================== Протокол выполнения ======================
```

```
section :: String -> IO ()
section t = printf "\n=== %s ===\n" t

check :: String -> Bool -> IO Bool
check label ok = do
  printf "  [%s] %s\n" (if ok then "OK  " else "FAIL" :: String) label
  return ok

studentLine :: Student -> String
studentLine s = printf "%-6s %-6s %-17s среднее %.2f"
  (studentName s) (studentGroup s) (show (studentGrades s)) (averageGrade s)

main :: IO ()
main = do
  hSetEncoding stdout utf8

  section "4.1. Перечисления: Day"
  mapM_ (\d -> printf "  %-10s номер %d  выходной: %s  следующий: %s\n"
                 (show d) (dayNumber d) (show (isWeekend d)) (show (nextDay d)))
        [minBound .. maxBound :: Day]
  printf "  Сравнение из Ord:   Monday < Friday = %s\n" (show (Monday < Friday))
  printf "  Диапазон из Enum:   [Monday .. Wednesday] = %s\n"
         (show [Monday .. Wednesday])

  section "4.2. Записи: Student"
  mapM_ (putStrLn . ("  " ++) . studentLine) students
  printf "  Лучший студент: %s (%.2f)\n"
         (studentName (topStudent students)) (averageGrade (topStudent students))
  let aliceFixed = updateGrade alice 4 5
  printf "  updateGrade alice 4 5: %s -> %s\n"
         (show (studentGrades alice)) (show (studentGrades aliceFixed))
  printf "  индекс вне диапазона:  %s\n"
         (show (studentGrades (updateGrade alice 99 2)))

  section "4.3. Сумма типов: Shape"
  let shapes = [Circle 2, Rectangle 3 4, Triangle 3 4 5]
  mapM_ (\sh -> printf "  %-22s площадь %8.4f  периметр %8.4f\n"
                  (show sh) (area sh) (perimeter sh)) shapes
  printf "  Triangle 1 2 10 корректен: %s\n" (show (isValidShape (Triangle 1 2
10)))

  section "4.4. Ручной экземпляр Eq: Color"
  let red  = RGB 255 0 0
      red' = RGB 255 0 0
      blue = RGB 0 0 255
  printf "  %s == %s -> %s\n" (show red) (show red') (show (red == red'))
  printf "  %s == %s -> %s\n" (show red) (show blue) (show (red == blue))
  printf "  /= получен бесплатно из определения класса: %s\n" (show (red /= blue))

  section "4.5. Рекурсивные типы: Tree"
  printf "  sampleTree  = %s\n" (show sampleTree)
  printf "  treeSize    = %d\n" (treeSize sampleTree)
  printf "  treeDepth   = %d\n" (treeDepth sampleTree)
  printf "  flattenTree = %s\n" (show (flattenTree sampleTree))
  printf "  mapTree (*10) = %s\n" (show (flattenTree (mapTree (* 10) sampleTree)))
  printf "  mapTree show  = %s\n" (show (flattenTree (mapTree show sampleTree)))

  section "Самопроверка"
  results <- sequence
    [ check "isWeekend Saturday" (isWeekend Saturday)
    , check "not (isWeekend Monday)" (not (isWeekend Monday))
    , check "nextDay Sunday == Monday" (nextDay Sunday == Monday)
    , check "nextDay Monday == Tuesday" (nextDay Monday == Tuesday)
    , check "dayNumber Sunday == 7" (dayNumber Sunday == 7)
    , check "averageGrade alice == 4.4" (abs (averageGrade alice - 4.4) < 1e-9)
    , check "topStudent == Кэрол" (studentName (topStudent students) == "Кэрол")
    , check "updateGrade заменяет оценку"
        (studentGrades (updateGrade alice 4 5) == [5,4,5,5,5])
    , check "updateGrade не портит исходную запись"
        (studentGrades alice == [5,4,5,5,3])
    , check "area (Circle 2) == 4*pi" (abs (area (Circle 2) - 4 * pi) < 1e-9)
```

```
  , check "area (Rectangle 3 4) == 12" (abs (area (Rectangle 3 4) - 12) < 1e-9)
  , check "area (Triangle 3 4 5) == 6" (abs (area (Triangle 3 4 5) - 6) < 1e-9)
  , check "perimeter (Triangle 3 4 5) == 12"
      (abs (perimeter (Triangle 3 4 5) - 12) < 1e-9)
  , check "RGB 255 0 0 == RGB 255 0 0" (RGB 255 0 0 == RGB 255 0 0)
  , check "RGB 255 0 0 /= RGB 0 0 255" (RGB 255 0 0 /= RGB 0 0 255)
  , check "treeSize sampleTree == 4" (treeSize sampleTree == 4)
  , check "treeDepth sampleTree == 3" (treeDepth sampleTree == 3)
  , check "flattenTree sampleTree == [1,2,3,4]"
      (flattenTree sampleTree == [1,2,3,4])
  , check "mapTree сохраняет форму"
      (treeSize (mapTree (* 2) sampleTree) == treeSize sampleTree)
  , check "mapTree (*10) даёт [10,20,30,40]"
      (flattenTree (mapTree (* 10) sampleTree) == [10,20,30,40])
  ]
printf "\nПройдено %d из %d проверок.\n"
       (length (filter id results)) (length results)
```