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

ОТЧЁТ

по лабораторной работе № 1

«Основы Haskell: типы, функции, pattern matching»

по дисциплине «Функциональное программирование»

Выполнил: студент группы ПМИМ-61

Резанко А. А.

Проверил: _____

Новосибирск, 2026

Лабораторная работа № 1

Основы Haskell: типы, функции, pattern matching

Цель работы

Освоить базовый синтаксис Haskell, систему типов, определение функций и сопоставление с образцом.

Теоретические сведения

Haskell – чисто функциональный язык со статической типизацией. Каждое выражение имеет тип, и этот тип выводится компилятором без указаний программиста. Функция задаётся как отображение аргументов в результат: одному и тому же набору аргументов всегда отвечает один и тот же результат, а изменяемого состояния и побочных эффектов в чистом коде нет.

Типы числовых литералов не фиксированы: выражение вроде суммы двух литералов получает тип с ограничением класса, и конкретный тип подбирается местом использования. Разбор случаев записывается тремя способами: условным выражением, сопоставлением с образцом по структуре аргумента и охраняющими условиями – набором логических условий при одном уравнении.

Средства и порядок выполнения

Компилятор GHC 9.4.7, операционная система Ubuntu 24.04. Исходный текст программы размещён в файле Lab1.hs и приведён полностью в приложении А. Сборка и запуск:

```
ghc -O0 -Wall Lab1.hs -o lab1
./lab1
```

Ключ -Wall включает полный набор предупреждений компилятора; сборка проходит без замечаний. В программу встроено 18 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

1.1 Простые выражения и типы

Команда :type сообщает тип выражения, не вычисляя его. Для числовых литералов интерпретатор показывает не конкретный тип, а ограничение класса: сумма двух литералов имеет тип с контекстом Num, то есть годится для любого числового типа, а выбор делается в месте применения.

Деление и целочисленное деление различаются именно классом типов. Операция деления объявлена в классе Fractional и на пятёрке и двойке даёт 2.5; функция div принадлежит классу Integral, отбрасывает дробную часть и даёт 2. Применить деление к значению типа Int компилятор не позволит: Int не является экземпляром Fractional.

Строка в Haskell – синоним списка символов, поэтому список из трёх литер имеет тип [Char] и печатается как строка.

Протокол сеанса GHCi:

```
ghci> :type 5 + 3
5 + 3 :: Num a => a
ghci> 5 + 3
8
ghci> :type 5 / 2
5 / 2 :: Fractional a => a
ghci> 5 / 2
2.5
ghci> :type div 5 2
div 5 2 :: Integral a => a
ghci> div 5 2
2
ghci> :type mod 17 3
mod 17 3 :: Integral a => a
ghci> mod 17 3
2
ghci> :type "hello" ++ " world"
"hello" ++ " world" :: [Char]
ghci> "hello" ++ " world"
"hello world"
ghci> :type ['a','b','c']
['a','b','c'] :: [Char]
ghci> ['a','b','c']
"abc"
ghci> :type True && False
True && False :: Bool
ghci> True && False
False
```

```
ghci> :type not (5 > 3)
not (5 > 3) :: Bool
ghci> not (5 > 3)
False
ghci> maxBound :: Int
9223372036854775807
ghci> product [1..20] :: Int
2432902008176640000
ghci> product [1..21] :: Int
-4249290049419214848
ghci> product [1..21] :: Integer
51090942171709440000
ghci> sqrt 2 :: Float
1.4142135
ghci> sqrt 2 :: Double
1.4142135623730951
```

Листинг:

```
-- Значения выражений из задания; их типы сняты отдельно в GHCi
-- (команда :type, протокол сеанса приведён в отчёте).
expressions :: [(String, String)]
expressions =
  [ ("5 + 3",                  show (5 + 3 :: Integer))
  , ("5 / 2",                  show (5 / 2 :: Double))
  , ("div 5 2",                show (div 5 2 :: Integer))
  , ("mod 17 3",               show (mod 17 3 :: Integer))
  , ("\"hello\" ++ \" world\"", show ("hello" ++ " world"))
  , ("['a','b','c']",          show ['a', 'b', 'c'])
  , ("True && False",          show (True && False))
  , ("not (5 > 3)",            show (not (5 > (3 :: Int))))
  ]
```

Результат работы:

```
5 + 3                    = 8
5 / 2                    = 2.5
div 5 2                  = 2
mod 17 3                 = 2
"hello" ++ " world"      = "hello world"
['a','b','c']            = "abc"
True && False            = False
not (5 > 3)              = False
5 / 2 и div 5 2 различаются: первое – деление в поле Double,
второе – целочисленное деление с отбрасыванием дробной части.
```

1.2 Определение функций

Сигнатуры выписаны явно, хотя компилятор вывел бы их сам: явная сигнатура фиксирует замысел и превращает описку в ошибку сборки, а не в неожиданный вывод типа.

Функция поиска большего значения имеет контекст Ord и работает для любого упорядоченного типа – в протоколе она применена и к числам, и к строкам. Факториал здесь записан условным выражением: базовый случай задан сравнением аргумента с единицей.

В площади круга показатель степени аннотирован типом Int: без аннотации ключ -Wall сообщает о неоднозначности типа числового литерала.

Листинг:

```
-- Площадь круга по радиусу
circleArea :: Double -> Double
circleArea r = pi * r ^ (2 :: Int)

-- Большее из двух значений любого упорядоченного типа
myMax :: Ord a => a -> a -> a
myMax x y = if x >= y then x else y

-- Факториал через if-then-else
factorialIf :: Integer -> Integer
factorialIf n = if n <= 1 then 1 else n * factorialIf (n - 1)
```

Результат работы:

```
circleArea 1    = 3.141593
circleArea 2.5  = 19.634954
myMax 3 7       = 7
myMax "abc" "abd" = "abd"
factorialIf 10  = 3628800
```

1.3 Сопоставление с образцом

Сопоставление с образцом разбирает аргумент по структуре. Уравнения просматриваются сверху вниз, поэтому базовые случаи стоят перед общим: если поставить общее уравнение первым, до остальных управление не дойдёт, и компилятор сообщит о перекрытых образцах.

В функции названия дня недели семь уравнений с числовыми образцами и завершающий шаблон подчёркивания. Без него вызов с номером вне отрезка от одного до семи завершился бы ошибкой неполного сопоставления; ключ -Wall предупреждает о таком пропуске ещё при сборке.

Обе версии факториала – условная и с образцами – сверены между собой на отрезке от нуля до пятнадцати отдельной проверкой.

Листинг:

```
-- Тот же факториал, записанный сопоставлением с образцом.
-- Образцы разбираются сверху вниз, поэтому базовые случаи стоят первыми.
factorial :: Integer -> Integer
factorial 0 = 1
factorial 1 = 1
factorial n = n * factorial (n - 1)

-- Название дня недели по номеру
dayName :: Int -> String
dayName 1 = "Понедельник"
dayName 2 = "Вторник"
dayName 3 = "Среда"
dayName 4 = "Четверг"
dayName 5 = "Пятница"
dayName 6 = "Суббота"
dayName 7 = "Воскресенье"
dayName _ = "Неизвестный день"
```

Результат работы:

```
factorial 0..10 = [1,1,2,6,24,120,720,5040,40320,362880,3628800]
factorial 20    = 2432902008176640000
dayName 1 = Понедельник
```

```
dayName 2 = Вторник
dayName 3 = Среда
dayName 4 = Четверг
dayName 5 = Пятница
dayName 6 = Суббота
dayName 7 = Воскресенье
dayName 8 = Неизвестный день
```

1.4 Кортежи

Кортеж разбирается образцом, который связывает все компоненты сразу, без обращения к отдельным селекторам.

В функции определения четверти первое уравнение ловит начало координат литеральным образцом, остальные случаи разобраны охраняющими условиями, а ветка otherwise покрывает точки на осях. Порядок важен: пара нулей подошла бы и под общий образец, поэтому частный случай стоит выше.

Листинг:

```
-- Сумма квадратов компонент тройки
sumOfSquares :: (Int, Int, Int) -> Int
sumOfSquares (x, y, z) = x ^ (2 :: Int) + y ^ (2 :: Int) + z ^ (2 :: Int)

-- Четверть координатной плоскости, в которую попадает точка
quadrant :: (Double, Double) -> String
quadrant (0, 0) = "Начало координат"
quadrant (x, y)
  | x > 0 && y > 0 = "I четверть"
  | x < 0 && y > 0 = "II четверть"
  | x < 0 && y < 0 = "III четверть"
  | x > 0 && y < 0 = "IV четверть"
  | otherwise      = "На оси"
```

Результат работы:

```
sumOfSquares (1,2,3)   = 14
sumOfSquares (3,4,12)  = 169
quadrant (2.0,3.0)    = I четверть
quadrant (-2.0,3.0)   = II четверть
quadrant (-2.0,-3.0)  = III четверть
quadrant (2.0,-3.0)   = IV четверть
quadrant (0.0,5.0)    = На оси
quadrant (0.0,0.0)    = Начало координат
```

1.5 Охраняющие условия

Наибольший общий делитель вычисляется алгоритмом Евклида: пока второй аргумент отличен от нуля, пара заменяется остатком от деления.

В методичке эта функция записана сопоставлением с образцом, хотя задание требует охраняющих условий, а в рекурсивном вызове стоит имя myGcd вместо myGCD – такой текст не компилируется. В работе обе неточности устранены.

Модуль в базовом случае нужен для отрицательных аргументов: наибольший общий делитель неотрицателен по определению. Результат сверен со стандартной функцией gcd на всех парах чисел из отрезка от минус двадцати до двадцати – это одна из встроенных проверок.

Листинг:

```
-- НОД по алгоритму Евклида. В методичке функция записана сопоставлением
-- с образцом, а в рекурсивном вызове опечатка в имени (myGcd); здесь
-- разбор случаев вынесен в охраняющие условия, как того требует задание.
myGCD :: Integer -> Integer -> Integer
myGCD a b
  | b == 0    = abs a
  | otherwise = myGCD b (a `mod` b)
```

Результат работы:

```
  myGCD   48   18 = 6
  myGCD   18   48 = 6
  myGCD  270  192 = 6
  myGCD   17    5 = 1
  myGCD  -48   18 = 6
  myGCD    7    0 = 7
```

Тестирование

Полный вывод программы за один запуск, включая встроенные проверки:

```
=== 1.1. Простые выражения ===
  5 + 3                    = 8
  5 / 2                    = 2.5
  div 5 2                  = 2
  mod 17 3                 = 2
  "hello" ++ " world"      = "hello world"
  ['a','b','c']            = "abc"
  True && False            = False
  not (5 > 3)              = False
  5 / 2 и div 5 2 различаются: первое – деление в поле Double,
  второе – целочисленное деление с отбрасыванием дробной части.

=== 1.2. Определение функций ===
  circleArea 1    = 3.141593
  circleArea 2.5  = 19.634954
  myMax 3 7       = 7
  myMax "abc" "abd" = "abd"
  factorialIf 10  = 3628800

=== 1.3. Pattern matching ===
  factorial 0..10 = [1,1,2,6,24,120,720,5040,40320,362880,3628800]
  factorial 20    = 2432902008176640000
  dayName 1 = Понедельник
  dayName 2 = Вторник
  dayName 3 = Среда
  dayName 4 = Четверг
  dayName 5 = Пятница
  dayName 6 = Суббота
  dayName 7 = Воскресенье
  dayName 8 = Неизвестный день

=== 1.4. Кортежи ===
  sumOfSquares (1,2,3)   = 14
  sumOfSquares (3,4,12)  = 169
  quadrant (2.0,3.0)    = I четверть
  quadrant (-2.0,3.0)   = II четверть
  quadrant (-2.0,-3.0)  = III четверть
  quadrant (2.0,-3.0)   = IV четверть
  quadrant (0.0,5.0)    = На оси
  quadrant (0.0,0.0)    = Начало координат

=== 1.5. Охраняющие условия ===
  myGCD   48   18 = 6
  myGCD   18   48 = 6
  myGCD  270  192 = 6
  myGCD   17    5 = 1
  myGCD  -48   18 = 6
  myGCD    7    0 = 7

=== Самопроверка ===
  [OK  ] circleArea 1 == pi
  [OK  ] myMax 3 7 == 7
  [OK  ] myMax работает на строках
  [OK  ] factorial 5 == 120
  [OK  ] factorial 0 == 1
  [OK  ] две версии факториала совпадают на 0..15
  [OK  ] dayName 3 == Среда
  [OK  ] dayName 0 уходит в шаблон подчёркивания
  [OK  ] sumOfSquares (1,2,3) == 14
  [OK  ] quadrant (2,3) == I четверть
  [OK  ] quadrant (-2,-3) == III четверть
  [OK  ] quadrant (0,5) == На оси
  [OK  ] quadrant (0,0) == Начало координат
  [OK  ] myGCD 48 18 == 6
  [OK  ] myGCD не зависит от порядка
```

```
  [OK  ] myGCD 7 0 == 7
  [OK  ] myGCD на отрицательных даёт модуль
  [OK  ] myGCD совпадает с gcd из Prelude

Пройдено 18 из 18 проверок.
```

Все 18 проверок пройдены (18 из 18), расхождений с ожидаемыми значениями нет.

Ответы на контрольные вопросы

1. Чем отличается Int от Integer? А Float от Double?

Тип Int – машинное целое фиксированной разрядности; в GHC на этой машине его наибольшее значение равно 9223372036854775807, то есть разрядность составляет 64 бита со знаком. При выходе за границу результат молча заворачивается по модулю: в протоколе GHCi произведение чисел от одного до двадцати одного в типе Int даёт отрицательное число −4249290049419214848.

Тип Integer – целое произвольной точности, ограниченное лишь памятью. То же произведение в Integer даёт верное значение 51090942171709440000. Плата за точность – скорость: арифметика Integer программная, а не аппаратная, поэтому Int берут там, где диапазон заведомо достаточен.

Типы Float и Double – числа с плавающей точкой одинарной и двойной точности, 32 и 64 бита. Различие видно на корне из двух: Float даёт 1.4142135, Double – 1.4142135623730951, то есть около семи значащих цифр против пятнадцати. Умолчание языка для дробных литералов – Double, и он же используется в работе.

2. Что такое полиморфизм в Haskell? Поясните на примере функции выбора максимума

Полиморфизм означает, что одно определение работает для целого семейства типов. В Haskell различают два вида. Параметрический полиморфизм не накладывает на тип никаких условий: функция длины списка работает для списка чего угодно, потому что сами элементы ей безразличны.

Полиморфизм с ограничением класса требует от типа определённого поведения. Функция выбора большего имеет контекст Ord: подойдёт любой тип, для которого определено сравнение. Реализация одна, а конкретная операция сравнения подставляется компилятором по типу в месте вызова –

поэтому в протоколе одна и та же функция сравнивает и числа, и строки, причём строки сравниваются лексикографически.

3. Как работает шаблон подчёркивания в сопоставлении с образцом?

Шаблон подчёркивания сопоставляется с любым значением и не связывает имени. Он используется там, где значение по существу не нужно: в завершающем уравнении, покрывающем все оставшиеся случаи, или на месте компоненты структуры, которая не участвует в вычислении.

Поскольку он совпадает с чем угодно, его место – последнее. Уравнение с подчёркиванием, поставленное выше остальных, перекроет их, и при ключе -Wall компилятор сообщит об этом. Из-за ленивости языка значение, попавшее под такой шаблон, не вычисляется вовсе.

4. В чём разница между условным выражением и охраняющими условиями?

Условное выражение – именно выражение, а не оператор: обе ветви обязательны и должны иметь один тип, потому что значение возвращается в любом случае. Ветвление на три и более случая приходится записывать вложенными условиями, и текст быстро уходит вправо.

Охраняющие условия – это набор логических условий при одном уравнении функции. Они проверяются сверху вниз, первое истинное определяет результат, а ключевое слово otherwise служит завершающей веткой. Запись остаётся плоской при любом числе случаев, условия могут пользоваться привязками из where и допустимы не только в уравнениях функции, но и в ветвях case. Если ни одно условие не выполнилось и завершающей ветки нет, вычисление прерывается ошибкой неполного сопоставления.

Выводы

В работе освоены базовый синтаксис Haskell и три способа разбора случаев: условное выражение, сопоставление с образцом и охраняющие условия. На одной и той же задаче – факториале – показано, что первые два способа дают совпадающие результаты, а выбор между ними определяется читаемостью.

Система типов выводит типы сама, но явные сигнатуры оказались полезны: они документируют замысел и превращают часть ошибок в ошибки сборки. Контекст класса в сигнатуре задаёт не конкретный тип, а требование к нему, за счёт чего одна функция обслуживает целое семейство типов.

Программа собирается без предупреждений при ключе -Wall и проходит все 18 встроенных проверок, включая сверку собственного алгоритма Евклида со стандартной функцией на 1681 паре значений.

ПРИЛОЖЕНИЕ А. Полный листинг программы (Lab1.hs)

```
-- Лабораторная работа № 1. Основы Haskell: типы, функции, pattern matching
-- Резанко А. А., ПМИМ-61
--
-- Сборка:  ghc -O0 -Wall Lab1.hs -o lab1
-- Запуск:  ./lab1

module Main where

import System.IO (hSetEncoding, stdout, utf8)
import Text.Printf (printf)

-- ===================== 1.1. Простые выражения и типы =====================

-- Значения выражений из задания; их типы сняты отдельно в GHCi
-- (команда :type, протокол сеанса приведён в отчёте).
expressions :: [(String, String)]
expressions =
  [ ("5 + 3",                  show (5 + 3 :: Integer))
  , ("5 / 2",                  show (5 / 2 :: Double))
  , ("div 5 2",                show (div 5 2 :: Integer))
  , ("mod 17 3",               show (mod 17 3 :: Integer))
  , ("\"hello\" ++ \" world\"", show ("hello" ++ " world"))
  , ("['a','b','c']",          show ['a', 'b', 'c'])
  , ("True && False",          show (True && False))
  , ("not (5 > 3)",            show (not (5 > (3 :: Int))))
  ]

-- ===================== 1.2. Определение функций =====================

-- Площадь круга по радиусу
circleArea :: Double -> Double
circleArea r = pi * r ^ (2 :: Int)

-- Большее из двух значений любого упорядоченного типа
myMax :: Ord a => a -> a -> a
myMax x y = if x >= y then x else y

-- Факториал через if-then-else
factorialIf :: Integer -> Integer
factorialIf n = if n <= 1 then 1 else n * factorialIf (n - 1)

-- ===================== 1.3. Pattern matching =====================

-- Тот же факториал, записанный сопоставлением с образцом.
-- Образцы разбираются сверху вниз, поэтому базовые случаи стоят первыми.
factorial :: Integer -> Integer
factorial 0 = 1
factorial 1 = 1
factorial n = n * factorial (n - 1)

-- Название дня недели по номеру
dayName :: Int -> String
dayName 1 = "Понедельник"
dayName 2 = "Вторник"
dayName 3 = "Среда"
dayName 4 = "Четверг"
dayName 5 = "Пятница"
dayName 6 = "Суббота"
dayName 7 = "Воскресенье"
dayName _ = "Неизвестный день"

-- ===================== 1.4. Кортежи =====================

-- Сумма квадратов компонент тройки
sumOfSquares :: (Int, Int, Int) -> Int
sumOfSquares (x, y, z) = x ^ (2 :: Int) + y ^ (2 :: Int) + z ^ (2 :: Int)

-- Четверть координатной плоскости, в которую попадает точка
```

```
quadrant :: (Double, Double) -> String
quadrant (0, 0) = "Начало координат"
quadrant (x, y)
  | x > 0 && y > 0 = "I четверть"
  | x < 0 && y > 0 = "II четверть"
  | x < 0 && y < 0 = "III четверть"
  | x > 0 && y < 0 = "IV четверть"
  | otherwise      = "На оси"

-- ==================== 1.5. Охраняющие условия ====================

-- НОД по алгоритму Евклида. В методичке функция записана сопоставлением
-- с образцом, а в рекурсивном вызове опечатка в имени (myGcd); здесь
-- разбор случаев вынесен в охраняющие условия, как того требует задание.
myGCD :: Integer -> Integer -> Integer
myGCD a b
  | b == 0    = abs a
  | otherwise = myGCD b (a `mod` b)

-- ==================== 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

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

  section "1.1. Простые выражения"
  mapM_ (\(e, v) -> printf "  %-24s = %s\n" e v) expressions
  printf "  5 / 2 и div 5 2 различаются: первое – деление в поле Double,\n"
  printf "  второе – целочисленное деление с отбрасыванием дробной части.\n"

  section "1.2. Определение функций"
  printf "  circleArea 1    = %.6f\n" (circleArea 1)
  printf "  circleArea 2.5  = %.6f\n" (circleArea 2.5)
  printf "  myMax 3 7       = %d\n" (myMax (3 :: Int) 7)
  printf "  myMax \"abc\" \"abd\" = %s\n" (show (myMax "abc" "abd"))
  printf "  factorialIf 10  = %d\n" (factorialIf 10)

  section "1.3. Pattern matching"
  printf "  factorial 0..10 = %s\n" (show (map factorial [0 .. 10]))
  printf "  factorial 20    = %d\n" (factorial 20)
  mapM_ (\n -> printf "  dayName %d = %s\n" n (dayName n)) [1 .. 7 :: Int]
  printf "  dayName 8 = %s\n" (dayName 8)

  section "1.4. Кортежи"
  printf "  sumOfSquares (1,2,3)   = %d\n" (sumOfSquares (1, 2, 3))
  printf "  sumOfSquares (3,4,12)  = %d\n" (sumOfSquares (3, 4, 12))
  mapM_ (\p -> printf "  quadrant %-12s = %s\n" (show p) (quadrant p))
        [(2, 3), (-2, 3), (-2, -3), (2, -3), (0, 5), (0, 0)]

  section "1.5. Охраняющие условия"
  mapM_ (\(a, b) -> printf "  myGCD %4d %4d = %d\n" a b (myGCD a b))
        [(48, 18), (18, 48), (270, 192), (17, 5), (-48, 18), (7, 0)]

  section "Самопроверка"
  results <- sequence
    [ check "circleArea 1 == pi" (abs (circleArea 1 - pi) < 1e-12)
    , check "myMax 3 7 == 7" (myMax (3 :: Int) 7 == 7)
    , check "myMax работает на строках" (myMax "abc" "abd" == "abd")
    , check "factorial 5 == 120" (factorial 5 == 120)
    , check "factorial 0 == 1" (factorial 0 == 1)
    , check "две версии факториала совпадают на 0..15"
        (map factorial [0 .. 15] == map factorialIf [0 .. 15])
    , check "dayName 3 == Среда" (dayName 3 == "Среда")
```

```
    , check "dayName 0 уходит в шаблон подчёркивания"
        (dayName 0 == "Неизвестный день")
    , check "sumOfSquares (1,2,3) == 14" (sumOfSquares (1, 2, 3) == 14)
    , check "quadrant (2,3) == I четверть" (quadrant (2, 3) == "I четверть")
    , check "quadrant (-2,-3) == III четверть" (quadrant (-2, -3) == "III
четверть")
    , check "quadrant (0,5) == На оси" (quadrant (0, 5) == "На оси")
    , check "quadrant (0,0) == Начало координат"
        (quadrant (0, 0) == "Начало координат")
    , check "myGCD 48 18 == 6" (myGCD 48 18 == 6)
    , check "myGCD не зависит от порядка" (myGCD 48 18 == myGCD 18 48)
    , check "myGCD 7 0 == 7" (myGCD 7 0 == 7)
    , check "myGCD на отрицательных даёт модуль" (myGCD (-48) 18 == 6)
    , check "myGCD совпадает с gcd из Prelude"
        (and [myGCD a b == gcd a b | a <- [-20 .. 20], b <- [-20 .. 20]])
    ]
  printf "\nПройдено %d из %d проверок.\n"
         (length (filter id results)) (length results)
```