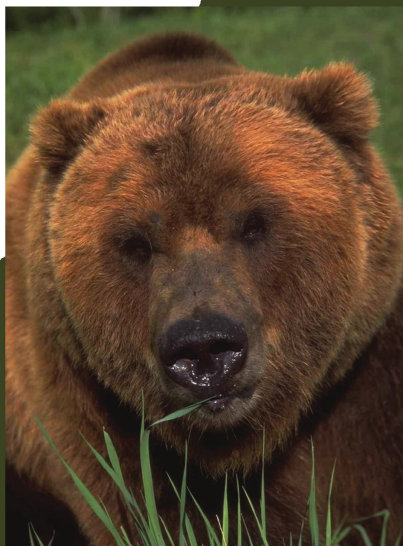


14 занимательных  
эссе о языке  
**Haskell**  
и функциональном  
программировании



Душкин Р. В.

Душкин Р. В.

# 14 занимательных эссе о языке Haskell и функциональном программировании

*2-е издание, электронное*

  
ИЗДАТЕЛЬСТВО  
Москва, 2023

УДК 004.4  
ББК 32.973.26-018.2  
Д86

**Душкин, Роман Викторович.**

Д86 14 занимательных эссе о языке Haskell и функциональном программировании / Р. В. Душкин. — 2-е изд., эл. — 1 файл pdf : 223 с. — Москва : ДМК Пресс, 2023. — Систем. требования: Adobe Reader XI либо Adobe Digital Editions 4.5 ; экран 10". — Текст : электронный.

ISBN 978-5-89818-621-0

В книге представлено 14 статей автора, которые в разное время были опубликованы или подготовлены к публикации в научно-популярном журнале для школьников и учителей «Потенциал». Статьи расположены и связаны таким образом, чтобы они представляли собой логически последовательное повествование от начал к более сложным темам. Также в книге сделан упор на практические знания, предлагается решение многих прикладных задач при помощи языка функционального программирования Haskell.

Книга будет интересна всем, кто живо интересуется функциональным программированием, студентам технических вузов, преподавателям информатики.

УДК 004.4  
ББК 32.973.26-018.2

**Электронное издание на основе печатного издания:** 14 занимательных эссе о языке Haskell и функциональном программировании / Р.В. Душкин. — Москва : ДМК Пресс, 2016. — 222 с. — ISBN 978-5-97060-360-4. — Текст : непосредственный.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

В соответствии со ст. 1299 и 1301 ГК РФ при устранении ограничений, установленных техническими средствами защиты авторских прав, правообладатель вправе требовать от нарушителя возмещения убытков или выплаты компенсации.

ISBN 978-5-89818-621-0

© Душкин Р. В., 2011  
© Оформление ДМК Пресс, 2016

### **Принимаются благодарности**

Вниманию всех читателей! Данная книга издана в электронном виде и распространяется абсолютно бесплатно. Вы можете свободно использовать её для чтения, копировать её для друзей, размещать в библиотеках на сайтах в сети Интернет, рассылать по электронной почте и при помощи иных средств передачи информации. Вы можете использовать текст книги частично или полностью в своих работах при условии размещения ссылок на оригинал и должном цитировании.

При этом автор будет несказанно рад получить читательскую благодарность, которая позволит как улучшить текст данной книги, так и более качественно подойти к подготовке следующих книг. Благодарности принимаются на счёт Яндекс.Деньги, на который можно перечислить малую лепту, и при помощи терминалов:

**4100137733052**

Убедительная просьба; при перечислении благодарности указывать в пояснении к переводу наименование книги или какое-либо иное указание на то, за что именно выражается благодарность.

# Оглавление

<b>От автора</b>	<b>8</b>
<b>Типовой процесс разработки программ на языке Haskell</b>	<b>10</b>
Инструментальные средства . . . . .	11
Описание процесса разработки . . . . .	14
<b>Функциональный подход в программировании</b>	<b>20</b>
Введение . . . . .	20
Общие свойства функций в функциональных языках программирования . . . . .	22
Примеры определения функций . . . . .	25
Заключение . . . . .	30
<b>Алгебраические типы данных в языке Haskell</b>	<b>32</b>
Введение . . . . .	32
Простые перечисления . . . . .	33
Параметризация . . . . .	37
Параметрический полиморфизм . . . . .	40
Заключение . . . . .	43
<b>Объектно-ориентированное и функциональное     программирование</b>	<b>44</b>
Введение . . . . .	44
Именованные поля и структуры . . . . .	46
Классы типов . . . . .	50
Экземпляры классов . . . . .	53

---

Окончательные замечания . . . . .	56
Заключение . . . . .	58
<b>Введение в <math>\lambda</math>-исчисление для начинающих</b>	<b>60</b>
Введение . . . . .	60
Неформальное описание теории . . . . .	62
Некоторые дополнения . . . . .	65
Редукция как стратегия вычислений . . . . .	66
Примеры кодирования данных и функций . . . . .	70
Заключение . . . . .	79
<b>Комбинаторы? — Это просто!</b>	<b>80</b>
Введение . . . . .	80
Формальная теория . . . . .	81
Примеры сложных комбинаторов . . . . .	84
Модуль на языке Haskell для преобразования комбинаторов . . . . .	88
Представление данных и функций . . . . .	91
Булевские значения . . . . .	91
Нумералы Чёрча . . . . .	92
Упорядоченные пары . . . . .	94
Общие замечания . . . . .	94
Заключение . . . . .	95
<b>Ввод и вывод на языке Haskell</b>	<b>97</b>
Введение . . . . .	97
Основы функционального ввода/вывода . . . . .	100
Стандартные функции ввода/вывода . . . . .	103
Примеры программ . . . . .	107
Вывод результатов исполнения функции на экран . . . . .	108
Альтернатива: экран или файл . . . . .	109
Копирование файлов . . . . .	111
Заключение . . . . .	113
<b>Простой интерпретатор команд</b>	<b>114</b>
Введение . . . . .	114
Постановка задачи . . . . .	115
Основной набор функций . . . . .	117

Вспомогательные типы данных . . . . .	117
Цикл интерпретации . . . . .	119
Функции для исполнения команд . . . . .	122
Заключение . . . . .	126
<b>Теория чисел и язык Haskell</b> . . . . .	<b>128</b>
Введение . . . . .	128
Простейшие задачи . . . . .	129
Такие непростые простые числа . . . . .	132
Числа Мерсенна . . . . .	135
Числа Ферма . . . . .	136
Числа Софи Жермен . . . . .	137
Другие последовательности простых чисел . . . . .	137
Совершенству нет предела . . . . .	138
Заключение . . . . .	141
<b>Магические квадраты и решение переборных задач</b> . . . . .	<b>142</b>
Введение . . . . .	142
Простейший вариант перебора . . . . .	144
Перебор с использованием перестановок . . . . .	148
Перебор с использованием размещений . . . . .	152
Дальнейшая универсализация алгоритма . . . . .	157
Заключение . . . . .	161
<b>Задача о ранце</b> . . . . .	<b>162</b>
Введение . . . . .	162
Классическая задача . . . . .	164
Реализация решения на языке Haskell . . . . .	166
Заключение . . . . .	171
<b>Кривая Дракона</b> . . . . .	<b>172</b>
Введение . . . . .	172
Что такое Кривая Дракона? . . . . .	176
Алгоритм построения . . . . .	178
Реализация на языке Haskell . . . . .	180
Подготовительные описания геометрических образов . . . . .	180
Построение Кривой Дракона . . . . .	184

---

Заключение . . . . .	187
<b>Немного о шахматных задачах</b>	<b>188</b>
Введение . . . . .	188
Вспомогательные программные сущности . . . . .	189
Задача о расстановке фигур . . . . .	193
Задача о ходе коня . . . . .	195
<b>Генерация рекурсивных сказок</b>	<b>199</b>
Введение . . . . .	199
Колобок . . . . .	200
Теремок . . . . .	207
Обобщение функций и построение генератора . . . . .	211
Репка . . . . .	216
Заключение . . . . .	219
<b>Литература</b>	<b>220</b>



# От автора

В период с 2006 по 2009 год в образовательном журнале для старшеклассников и учителей «Потенциал» (ISSN 1814-6422) были опубликованы или подготовлены к публикации четырнадцать научно-популярных статей о функциональном программировании и языке Haskell. Многие из данных статей нашли определённый отклик в сердцах и умах читателей, поэтому автором было решено свести их в одну книгу, выстроив в более или менее стройную систему повествования.

Вместе с тем со времени начала публикации научно-популярных статей прошло достаточное количество времени, чтобы и язык Haskell, и функциональное программирование в целом эволюционировали, поэтому необходима и актуализация информации, приведение её в более современный вид. Всё это значит, что в настоящей книге исходные статьи не только не приведены в исходном порядке публикации, но и в некоторых случаях достаточно серьёзно переработаны.

Книга будет полезна школьникам старших классов, живо интересующимся информатикой и смежными областями науки и техники, а также учителям, которые проводят факультативные занятия по информатике и компьютерной грамотности. Кроме того, я надеюсь, что книга также будет полезна студентам технических высших учебных заведений, которые хотели бы овладеть пониманием функционального программирования в целом и языка Haskell в частности. Ну и в любом случае книга станет неплохим источником идей, задач и их решений для всех, кто интересуется функциональным программированием.

Я выражаю самую серьёзную благодарность всем моим коллегам, которые в той или иной мере способствовали появлению многих из приведённых в книге статей на свет. Особо стоит отметить Антонюка Д. А. и Астапова Д. Е. (в том

числе и в качестве рецензента), чьи бесменные советы и предложения по улучшению всегда приходились кстати. Также хочу выразить благодарность всему коллективу редакции журнала «Потенциал» за его самоотверженный труд на поприще просвещения и популяризации науки среди подрастающего поколения. Отдельно хотелось бы отметить Ворожцова А. В., редактора отдела «Информатика».

Буду крайне признателен любым конструктивным комментариям, замечаниям и предложениям, которые можно присылать по адресу электронной почты `roman.dushkin@gmail.com`. Также по этому адресу можно присылать запросы на файлы с исходными кодами примеров, приведённых в книге (указывайте, пожалуйста, наименования эссе, для которых необходимо выслать исходные коды примеров).

В добрый путь.

*Душкин Р. В.  
Москва, 2011.*

# Типовой процесс разработки программ на языке Haskell

*Статья была подготовлена к публикации в один из номеров журнала «Потенциал» в конце 2009 года. Опубликована не была.*

*В эссе рассказывается о типовом процессе разработки программных средств на функциональном языке программирования Haskell. Описываются некоторые из имеющихся на текущий момент программных средств и инструментов, делающих процесс разработки простым и быстрым. Дается краткая суммарная информация о том, где и на каких условиях можно получить весь необходимый для работы инструментарий.*

В дальнейшем изложении в настоящей книге приводятся разнообразнейшие задачи из области математики и информатики, а также описываются решения этих задач при помощи языка функционального программирования Haskell. Опыт общения автора с читателями показал, что имеется проблема непосредственно практического характера — многие читатели просто не знают, что делать с приводимыми примерами и исходными кодами: какой инструментарий использовать, как компилировать программы, как загружать дополнительные пакеты и т. д.

Осмысление проблемы привело к пониманию того, что язык Haskell при всех его достоинствах очень сложно входит в русло практического использования как любителями, так и профессиональными программистами. В связи с этим возникает необходимость в использовании новой парадигмы популяризации языка Haskell и функционального подхода в программировании (впервые введена в книге [8]).

Данная парадигма предполагает необходимость всемерного распространения знаний и информации о практических средствах и методах разработки, а теоретическая часть может быть получена заинтересованными людьми самостоятельно, благо в литературе именно по теоретической части недостатка нет (см. в том числе книги [6, 7, 14, 15, 16]). Этот шаг позволит на первоначальном этапе заинтересовать и через интерес привлечь к функциональному программированию множество неопитов.

Данное эссе описывает типовой процесс разработки программного средства на языке Haskell на примере простейшего приложения «Hello, world!». Сложность разрабатываемого программного средства здесь совершенно не имеет значения, поскольку обрисовываются именно процесс и инструментарий. Читатель волен самостоятельно применять описываемые в разделе средства для решения произвольных задач, хотя бы и для рассмотрения примеров в следующих эссе, входящих в состав книги.

## Инструментальные средства

Весь набор инструментальных средств, используемых в работе над любым проектом, можно разделить на классы. Это относится не только к языку программирования Haskell, но и к любому другому языку программирования, впрочем, как и к любым иным проектам — от проектирования детской игрушки до постройки города. В принципе, разделить на классы инструментальные средства для работы с языком Haskell можно так, как показано на рис. 1.

Также инструментарий можно классифицировать по степени полезности и необходимости. Некоторые классы инструментов обычно не используются (препроцессоры для языка Haskell используются редко), а без, к примеру, трансляторов языка вообще невозможно обойтись. Так что ниже в таблице перечислены наиболее часто используемые инструментальные средства, которые подходят для каждого разработчика, для любого проекта.

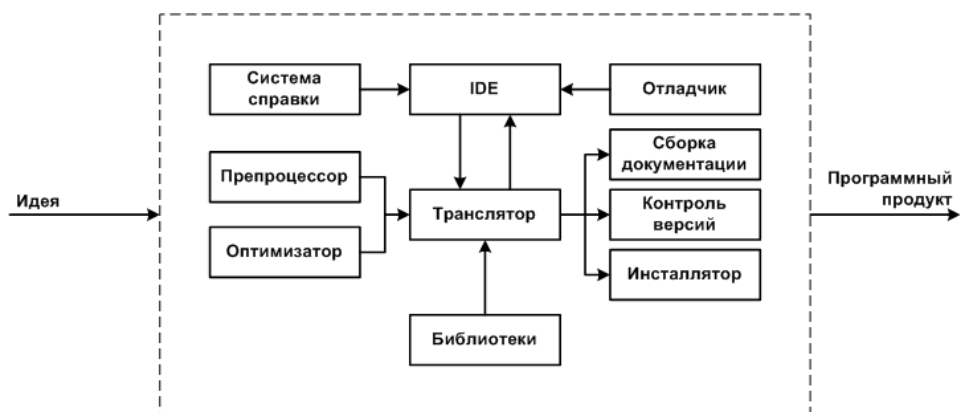


Рис. 1. Кибернетическая схема преобразования идеи в законченное программное средство

Инструмент	Описание
1	2
Компилятор GHC	Самый мощный и совершенный компилятор, на сегодняшний день не имеющий аналогов, — GHC. Реализует не только стандарт Haskell-98, но и множество расширений языка, многие из которых уже стали стандартом де-факто. Имеет возможность работы в режиме интерпретации (в состав поставки входит интерпретатор GHCi).  <a href="http://www.haskell.org/ghc/">http://www.haskell.org/ghc/</a>
Тестирование QuickCheck	Для проведения тестирования обычно используются специальные библиотеки вроде QuickCheck. Они позволяют создавать семантические правила, описывающие поведения функций в проекте, после чего запускать эти правила на проверку с различными аргументами. Проверка осуществляется компилятором в процессе сборки программы.

Продолжение на следующей странице

1	2
Документирование Haddock	<p data-bbox="492 331 984 358"><a href="http://www.md.chalmers.se/~rjmh/QuickCheck/">http://www.md.chalmers.se/~rjmh/QuickCheck/</a></p> <p data-bbox="492 366 1112 666">При написании исходных кодов можно сразу документировать программные сущности таким образом, что использование утилиты Haddock позволит собрать справочную систему для разработанного проекта в формате HTML. Это — правильный подход к разработке, поскольку документация позволяет без проблем передавать разработанный проект.</p> <p data-bbox="492 719 845 746"><a href="http://www.haskell.org/haddock/">http://www.haskell.org/haddock/</a></p>
Контроль версий Darcs	<p data-bbox="492 754 1112 975">Для хранения файлов с исходными кодами и ведения их версий можно воспользоваться утилитой распределённого хранения Darcs. Она обладает рядом дополнительных возможностей, превышающих стандартную функциональность систем контроля версий.</p> <p data-bbox="492 1028 732 1054"><a href="http://www.darcs.net/">http://www.darcs.net/</a></p>
Система сборки Cabal	<p data-bbox="492 1063 1112 1363">Утилита Cabal позволяет собрать пакет для распространения среди разработчиков и пользователей. Пакет собирается в стандартном формате, который может быть использован на произвольной платформе, поддерживающей язык Haskell. Кроме того, эта же утилита может установить в систему требуемый пакет, самостоятельно скачав его из сети Интернет.</p> <p data-bbox="492 1416 820 1442"><a href="http://www.haskell.org/cabal/">http://www.haskell.org/cabal/</a></p>

*Продолжение на следующей странице*

1	2
Распространение Hackage	Для того чтобы каждый желающий смог воспользоваться разработанным пакетом, его можно положить в общий архив исходных кодов на языке Haskell Hackage. Этот архив является централизованным, и сегодня в нём хранятся сотни различных пакетов для решения практически любых задач.  <a href="http://hackage.haskell.org/">http://hackage.haskell.org/</a>

Перечисленные в приведённой таблице инструментальные средства являются свободно распространяемым программным обеспечением, доступным для получения с указанных адресов в сети Интернет. Читателям настоятельно рекомендуется скачать перечисленные программные средства для возможности реальной работы с языком Haskell, что позволит без проблем работать над примерами, описанными далее в этой книге.

## Описание процесса разработки

После установки на рабочий компьютер перечисленного инструментария можно начинать работу над проектами. Ниже представлена типовая структура проекта на примере простейшего приложения типа «Hello, World!». Типовой проект на языке Haskell будет содержать следующие компоненты:

- 1) `_darcs` — каталог для хранения версий файлов с исходными кодами;
- 2) `Hello.hs` — главный файл проекта (в нём содержится функция `main`);
- 3) `Hello.cabal` — описание проекта для системы Cabal;
- 4) `Setup.hs` — сценарий сборки проекта в системе Cabal;
- 5) `README` — файл с информацией о проекте;
- 6) `LICENSE` — файл с описанием лицензии.

Конечно, каждый разработчик вправе вносить в указанную структуру изменения и дополнения, отражающие суть соответствующего проекта (например, файлы с исходными кодами хорошо бы разместить в подкаталоге `/src`). Здесь приведён именно пример простейшего проекта, состоящего, по сути, из одного файла с исходными кодами. К одному файлу (в примере — `Hello.hs`) добавляются дополнительные файлы и каталоги, необходимые для поддержания инфраструктуры проекта.

Следующая инструкция показывает пошаговое создание проекта «Hello, World!» до получения исполняемого файла и размещения его в архиве проектов.

- 1) Создание каталога проекта и файла с исходным кодом в нём. Содержимое файла может быть следующим:

```
module Main
```

```
— | Функция 'main', выводящая на экран приветствие.
```

```
main :: IO ()
```

```
main = putStr "Hello, World!"
```

- 2) Сохранение информации в системе хранения версий. Это делается при помощи следующих команд:

```
darcs init
```

```
darcs add Hello.hs
```

```
darcs record
```

При выполнении последней команды утилита Darcs задаст пользователю ряд вопросов, после чего файл `Hello.hs` будет сохранён в архиве для сохранения первой версии.

- 3) Поскольку файл с исходными кодами готов (само собой разумеется, что в случае сложных проектов необходимо быть уверенным в полной готовности всех файлов проекта), необходимо создать файл с описанием проекта для системы Cabal. Содержимое файла `Hello.cabal` содержит примерно следующие данные:

```
Name:                hello
```

```
Version:             0.0.0.1
```



```

Description:      Simplest Haskell Application
License:         GPL3
License-file:    LICENSE
Author:         John Smith
Maintainer:      john.smith@example.com
Build-Type:     Simple
Cabal-Version:   >= 1.8
Executable haq
  Main-is:       Hello.hs
  Build-Depends: base >= 3 && < 5

```

Если разрабатываемый проект зависит от каких-либо иных пакетов, то все они должны быть указаны в файле `.cabal` для сборки в поле `Build-Depends`.

- 4) Далее необходимо написать сценарий сборки. Для подавляющего числа проектов сценарий `Setup.hs` одинаков (утилита Cabal позволяет использовать как обычные файлы `.hs`, так и коды в литературном стиле `.lhs`). Его содержимое следующее:

```
import Distribution.Simple
```

```
main = defaultMain
```

- 5) Если есть потребность, можно разработать файлы `README` с общей информацией о проекте и `LICENSE` с информацией о лицензировании проекта. Эти файлы не участвуют в процессе сборки, но предназначаются для более целостного представления проекта потенциальным потребителям.
- 6) Новые файлы также необходимо сохранить в системе хранения версий `Darcs`. Это делается при помощи выполнения следующих команд (опять же, при их выполнении утилита может задать дополнительные вопросы, на которые необходимо ответить по существу):

```
darcs add Hello.cabal Setup.hs LICENSE README
darcs record --all
```

- 7) Сборка проекта может осуществляться как при помощи транслятора, так и при помощи утилиты Cabal. Последний способ гарантирует то, что получаемые исполняемые файлы проекта будут готовы для сохранения в централизованном архиве Hackage. Сборка проекта осуществляется при помощи выполнения следующих команд:

```
cabal install --prefix=\$HOME --user
```

Выполнение команды создаст в подкаталоге `/bin` исполняемый файл проекта. Его можно будет запускать на исполнение.

- 8) Теперь можно сгенерировать документацию для проекта. Это также можно осуществить как при помощи утилиты Haddock, так и непосредственно используя систему Cabal, поскольку имеется централизованная интеграция инструментов с этой системой сборки. Документация генерируется следующей командой:

```
cabal haddock
```

В результате исполнения команды будет сгенерирован набор файлов `.html` и некоторых дополнительных к ним служебных файлов и подкаталогов. Данные файлы содержат документацию проекта, созданную на основании комментариев разработчика, написанных в исходных кодах.

- 9) Также в проект можно добавить методы автоматизированного тестирования функций при помощи библиотеки QuickCheck. Организацию тестирования можно осуществить при помощи системы хранения версий Darcs при сохранении очередных изменений — утилита самостоятельно будет осуществлять тестирование и будет производить уведомление пользователя в случаях, если тестирование завершилось неудачей.

В настоящем примере функция `main` слишком проста для проведения тестирования. В более сложных проектах желательно организовывать автоматизированное тестирование для «отлова» потенциальных логических ошибок. Пусть сценарий тестирования расположен в файле `Test.hs`, в этом случае интеграция процесса тестирования с процессом сохранения версии в системе хранения выполняется при помощи следующей команды:

```
darcs setpref test "runhaskell Tests.hs"
```

Само собой разумеется, что новый файл `Test.hs` также необходимо сохранить в системе хранения версий. Кстати, в последних версиях Cabal тестирование можно проводить средствами этой утилиты.

- 10) После появления стабильной версии проекта (версии, которая не содержит критических ошибок, приводящих к «падению» приложения) её можно пометить специальным образом в системе хранения в целях быстрого доступа к файлам версии. Это делается при помощи следующей команды:

```
darcs tag
```

Утилита Darcs задаст дополнительные вопросы о том, какая метка должна быть присвоена текущим версиям файлов в репозитории.

- 11) Далее при помощи системы Cabal необходимо создать пакет для распространения и помещения в централизованный архив Hackage. Перед этим желательно проверить, всё ли готово для загрузки в централизованный архив, для чего можно выполнить такую команду:

```
cabal check
```

Если всё в порядке, то можно готовить файл для загрузки в Hackage. Это делается просто, при помощи команды:

```
cabal sdist
```

В результате выполнения команды в каталоге проекта будет создан файл `hello-0.0.0.1.tar.gz`, в котором находятся полное описание проекта, исходные коды для него, а сам проект будет полностью готов для размещения в архиве Hackage.

- 12) Окончательным шагом в работе над проектом будет отсылка файла с дистрибутивом (пакетом) в централизованный архив Hackage. Для этого необходимо иметь учётную запись пользователя архива (её можно получить при помощи регистрации на веб-сайте <http://hackage.haskell.org/>). После входа в архив под учётной записью будет доступна форма для загрузки файла. Загрузка файла

пакета на веб-сайт приведёт к автоматической проверке и, в случае её успешности, помещению пакета в архив.

Дело сделано. Но для более серьёзного проекта будет хорошим тоном создание на веб-сайте <http://www.haskell.org/> страницы с описанием проекта. Данный сайт — официальный сайт сообщества программистов на языке Haskell. Он использует технологию Wiki, при помощи которой любой желающий посетитель может внести в страницы веб-сайта информацию. Создание новых разделов с описаниями проектов поощряется владельцами сайта, и сегодня на нём содержится описание большинства существующих проектов, созданных как на языке Haskell, так и для работы с ним.

Таков общий процесс разработки проектов на языке Haskell. Автор искренне надеется, что данное краткое эссе станет хорошим подспорьем как для начинающих программистов, так и для умелых разработчиков. Собственно, заниматься со всеми последующими разделами книги можно именно по приведённой схеме.

Также необходимо отметить, что в 2009 году (уже после написания этого эссе) сообществом языка Haskell был подготовлен целостный пакет прикладного программного обеспечения для работы на этом языке Haskell Platform. Данный пакет включает в себя компилятор GHC, систему генерации справки Haddock, систему подготовки дистрибутивов Cabal, пакет тестирования QuickCheck, а также несколько других часто используемых в работе пакетов и утилит.

Актуальное описание того, как разрабатывать программы на языке Haskell, готовить их к распространению и помещать в централизованный архив Hackage, всегда можно найти по адресу [http://www.haskell.org/haskellwiki/How\\_to\\_write\\_a\\_Haskell\\_program](http://www.haskell.org/haskellwiki/How_to_write_a_Haskell_program).

# Функциональный подход в программировании

*Статья опубликована в № 08 (56) журнала «Потенциал» в августе 2009 года. Сама по себе статья является упрощённой переработкой статьи «Функции и функциональный подход», опубликованной в № 01 научно-практического журнала «Практика функционального программирования».*

*В эссе в сжатой форме рассказывается про функциональный подход к описанию вычислительных процессов (и в общем к описанию произвольных процессов в реальном мире), а также про применение этого подхода в информатике в функциональной парадигме программирования. Примеры реализации функций даются на языке программирования Haskell.*

## Введение

Имеется несколько различающихся подходов к организации вычислительных процессов, наиболее известными и в какой-то степени «конкурирующими» являются императивный (процедурный) и функциональный стили. Эти стили вычислений были известны в далёком прошлом, и сейчас уже невозможно узнать, какой подход был разработан первым.

Последовательности шагов вычислений, как особенность процедурного стиля, можно рассматривать в качестве естественного способа выражения человеческой деятельности при её планировании. Это связано с тем, что человеку приходится жить в мире, где неумолимый бег времени

и ограниченность ресурсов каждого отдельного индивидуума заставлял людей планировать по шагам свою дальнейшую жизнедеятельность.

Вместе с тем нельзя сказать, что функциональный стиль вычислений был неизвестен человеку, а появился только с возникновением теории вычислений в том или ином виде в конце XIX — начале XX века. Декомпозиция задачи на подзадачи и выражение ещё нерешённых проблем через уже решённые — эти методики также были известны с давних времён, а именно они составляют суть функционального подхода. Именно этот подход и является предметом рассмотрения настоящего раздела, а объясняться его положения будут при помощи функционального языка Haskell (для изучения языка можно воспользоваться книгой [6]).

Несмотря на то что фактически функциональный подход к вычислениям был известен с давних времён, его теоретические основы стали разрабатываться вместе с началом работ над вычислительными машинами — сначала механическими, а потом уже и электронными. С развитием формальной логики и обобщением множества сходных идей под сводом кибернетики появилось понимание того, что функция является прекрасным математическим формализмом для описания реализуемых в физическом мире устройств [2]. Но не всякая функция, а только такая, которая обладает рядом важных свойств, а именно: во-первых, она оперирует исключительно своей внутренней памятью, а во-вторых, она детерминирована (то есть её значение зависит только от входных параметров — это свойство будет подробно рассмотрено далее). Данные ограничения на реализуемость в реальности связаны с физическими законами сохранения, в первую очередь энергии. Именно такие «чистые» процессы рассматриваются в кибернетике при помощи методологии чёрного ящика — результат работы такого ящика зависит только от значений входных параметров.

Классическая иллюстрация, демонстрирующая эту ситуацию, встречается в большинстве учебников по кибернетике и смежным дисциплинам:

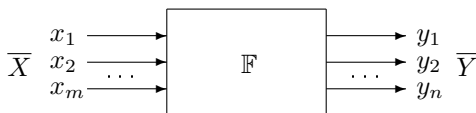


Рис. 2. Чёрный ящик функции  $F$  с вектором входов  $\bar{X}$  и вектором выходов  $\bar{Y}$