Рабочая группа
разработки R

Определение языка R

Версия 3.5.2 (2018-12-20)
DRAFT

Александр Фоменко

Определение языка R. Версия 3.5.2 (2018-12-20) DRAFT

«Издательские решения»

Фоменко А. А.

Определение языка R. Версия 3.5.2 (2018-12-20) DRAFT / А. А. Фоменко — «Издательские решения»,

ISBN 978-5-44-966029-9

Данная книга является переводом одноименной книги из комплекта технической документации, поставляемой в составе дистрибутива R, и призвана восполнить пробел в русской локализации системы R.

ISBN 978-5-44-966029-9

Содержание

Определение языка R
Версия 3.5.2 (2018-12-20) DRAFT

Версия 3.5.2 (2018-12-20) DRAFT
Рабочая группа разработки R

Переводчик Александр Александрович Фоменко

ISBN 978-5-4496-6029-9
Создано в интеллектуальной издательской системе Ridero

R Language Definition

Version 3.5.2 (2018-12-20) DRAFT
R Development Core Team

По вопросам перевода обращаться по адресу:
http://www.aafomenko@yandex.ru

1. Введение

R – система для статистического вычисления и графики. Она включает, между прочим, язык программирования, высокоуровневую графику, интерфейсы к другим языкам и средства отладки. Этот справочник детализирует и определяет язык ***R.***

Язык ***R*** – диалект ***S,*** который был разработан в 1980-ых и с тех пор находится в широком использовании в статистическом сообществе. Его основной разработчик, Джон М. Чемберс, был награжден Премией по системному программному обеспечению ACM 1998 года за S.

У синтаксиса языка есть поверхностное подобие с ***C,*** но семантика принадлежит к семейству FPL (языкам функционального программирования) с более сильной аффилированностью с Lisp и APL. В частности, возможно «вычислять на языке», который поочередно позволяет записать функции, которые берут выражения в качестве входа, что-то, что часто полезно для статистического моделирования и графики.

Можно получить довольно отдаленное использование ***R*** в интерактивном режиме, выполняя простые выражения из командной строки. Некоторые пользователи никогда, возможно, не уйдут с этого уровня, другие захотят написать свои собственные функции или оперативно систематизировать однообразную работу или на перспективу записать дополнительные пакеты для новой функциональности.

Цель этого справочника состоит в документировании языка по существу. Это означает, объекты показаны, так как они работают, и детали процесса вычисления выражений, которые полезно знать при программировании функций ***R.*** Главные подсистемы для определенных задач, таких как графика, описаны только кратко в этом справочнике и будут задокументированы отдельно.

Хотя большая часть текста одинаково применима к ***S,*** есть также некоторые существенные различия, и чтобы не перепутать проблему, сконцентрируемся на описании ***R.***

Проект языка содержит много тонкостей и распространенных ошибок, которые могут удивить пользователя. Большинство из них происходит из-за соображений непротиворечивости на более глубоком уровне, как мы объясним. Есть также много полезных ярлыков и идиом, которые позволяют пользователю выражать вполне сложные операции кратко. Многие из них становятся естественными по мере ознакомления с базовыми понятиями. В некоторых случаях существует много способов выполнить задачу, но некоторые из методов основаны на реализации языка, а другие работают на более высоком уровне абстракции. В таких случаях укажем на преимущественное использование.

Предполагается некоторое знакомство с ***R.*** Данное руководство не введение в ***R,*** а скорее справочник программиста. Другие справочники предоставляют дополнительную информацию: в особенности раздел «Предисловие» во Введении в ***R*** предоставляет введение в ***R,*** и раздел «Система и интерфейсы внешних языков» в Написание расширений ***R*** детализирует расширение ***R,*** используя скомпилированный код.

2. Объекты

На каждом машинном языке переменные обеспечивают средство доступа к данным, хранящимся в памяти. ***R*** не обеспечивает прямой доступ к памяти компьютера, а скорее обеспечивает много специализированных структур данных, именуемых как объекты. Эти объекты упомянуты через символы или переменные. В ***R,*** однако, символы – самостоятельно объекты и могут управляться таким же образом как любой другой объект. Этим он отличается от многих других языков и имеет широко распространяющиеся следствия.

В этой главе даны предварительные описания различных структур данных, предоставленных в ***R.*** Более детальные обсуждения многих из них будут найдены в последующих главах. Функция определения *typeof* в ***R*** возвращает *тип* объекта ***R.*** Заметим, что в коде ***C,*** лежащем в основе ***R,*** все объекты являются указателями на структуру с определением типа SEXPREC; различные типы данных ***R*** представлены в ***C*** SEXPTYPE, который определяет, как используется информация в различных частях структуры.

Следующая таблица описывает возможное значение, возвращенное *typeof*, и их значение.

«NULL»

NULL

«symbol»

имя переменной

«pairlist»

парный объект (в основном внутренний)

«closure»

функция

«environment»

окружающая среда

«promise»

объект, используемый для отложенной оценки

«language»

*конструкция языка **R***

«special»

внутренняя функция, которая не вычисляет свои аргументы

«builtin»

внутренняя функция, которая вычисляет свои аргументы

«char» а «scalar»

*строковый объект (только внутренний) ****

«logical»

вектор, содержащий логические значения

«integer»

вектор, содержащий целые значения

«double»

вектор, содержащий реальные значения

«complex»

вектор, содержащий комплексные значения

«character»

вектор, содержащий символьные значения

«…»

*аргумент определенной переменной длины ****

«any»

специальный тип, который заменяет все типы: не существует объектов такого типа

«expression»

объект выражение

«list»

список

«bytecode»

*код в байтах (только внутренне) ****

«externalptr»

объект внешнего указателя

«weakref»

объект слабой ссылки

«raw»

вектор, содержащий байты

«S4»

объект S4, который не является простым объектом

Пользователи не могут просто получить объекты, помеченные «***».

Функциональный режим дает информацию о режиме объекта в смысле Becker, Chambers & Wilks (1988), и является более совместимым с другими реализациями языка ***S***. Наконец, функция *storage.mode* показывает режим хранения ее аргумента в смысле Беккера и др. (1988). Она обычно используется при вызове функции, записанной на другом языке, таких как ***C*** или ФОРТРАН для гарантирования, что объекты ***R*** имеют тип данных, который ожидает вызывае-

мая подпрограмма. (На языке ***S*** векторы с целочисленными или действительными значениями имеют оба «числовой» режим, таким образом, их режимы хранения нужно отличать.)

```
> x <– 1:3
> typeof (x)
[1] «integer»
> mode (x)
[1] «numeric»
> storage.mode (x) [1] «integer»
```

Объекты в ***R*** часто преобразовываются к различным типам во время вычислений. Также имеется много доступных функций для выполнения явного преобразования. При программировании на языке ***R*** тип объекта обычно не влияет на вычисления, однако, имея дело с внешними языками или операционной системой, часто необходимо гарантировать корректность типа объекта.

2.1. Основные типы

2.1.1. Векторы

Вектора рассматриваются как непрерывная последовательность ячеек, содержащих данные. Доступ к ячейкам осуществляется через операции индексирования, такими, как x [5]. Более детально рассмотрено в разделе 3.4 [индексирование].

R имеет шесть основных («атомарных») типов векторов: *logical, integer, real, complex, string (or character)* и *raw*. Режим и режим хранения для разных типов векторов перечислены в следующей таблице.

Тип	mode	storage.mode
logical	*logical*	*logical*
integer	*numeric*	*integer*
double	*numeric*	*double*
complex	*complex*	*complex*
character	*character*	*character*
raw	*raw*	*raw*

Отдельные числа, такие как 4.2, и строка, такая как «*four point two*», все еще векторы, длины 1; нет больше основных типов. Возможны (и полезны) векторы с нулевой длиной.

У векторов строки есть режим и режим хранения «*character*». Отдельный элемент символьного вектора часто упоминается как *символьная строка*.

2.1.2. Списки

Списки («универсальные векторы») являются другим видом хранения данных. У списков есть элементы, каждый из которых может содержать любой тип объекта ***R,*** то есть элементы списка не обязательно имеют одинаковый тип. К элементам списка получают доступ посредством трех различных операций индексации. Они объяснены подробно в Разделе 3.4 [Индексирование].

Списки – векторы, и основные типы векторов упоминаются как атомарные векторы, где необходимо исключить списки.

2.1.3. Языковые объекты

Есть три типа объектов, которые составляют язык ***R,*** а именно: *call* (вызов), *expressions* (выражения) и *name* (имя). Так как у ***R*** есть объекты типа «выражение», то попытаемся избежать использования слова «выражение» в других контекстах. В определенных синтаксически корректных высказываниях выражения будут упоминаться как *операторы*.

У этих объектов есть режимы «*call», «expression»* и «*name*», соответственно.

Они могут быть созданы непосредственно из выражений, используя механизм кавычек и преобразованы «в» и «из» списков функциями *as.call* и *as.list*. Могут быть извлечены компоненты дерева синтаксического анализа, используя стандартные операции индексации.

2.1.4. Символьные объекты

Символы обращаются к объектам ***R.*** Обычно имя любого объекта ***R*** – символ. Символы могут быть созданы через функции *as.name* и кавычку.

Символы имеют режим «*name*», режим хранения «*symbol*» и тип «*symbol*». Они могут быть преобразованы «в» и «из» символьных строк, используя *as.character* и *as.name*. Они естественно появляются как атомы проанализированных выражений, попробуй, например, *as.list (quote (x + y)).*

2.1.5. Выражения – объекты

В ***R*** можно иметь объекты типа» *expression*». Выражение содержит одно или более предложений. Оператор – синтаксически корректный набор маркеров. Объекты выражения – специальные объекты языка, которые содержат проанализированные, но неоцененные операторы ***R.*** Основное различие состоит в том, что объект выражения может содержать несколько таких выражений. Другие более тонкие различия состоят в том, что объекты типа «*expression*» оцениваются лишь при явной передаче на вычисление, тогда как другие объекты языка могут быть оценены в некоторых неожиданных случаях.

Объект выражения ведет себя также как список, и к его компонентам можно получить доступ таким же образом как компонентам списка.

2.1.6 Объекты функции

В ***R*** функции – объекты и могут управляться почти таким же способом как любой другой объект. У функций (или более точно, обертка функции) есть три основных компонента: формальный список аргументов, тело и окружающая среда. Список аргументов – список разделенных запятой значений аргументов. Аргумент может быть символом, или конструкцией «*symbol = default*», или специальным аргументом «…'. Вторая форма аргумента используется для указания значения по умолчанию для аргумента, которое будет использоваться при вызове функция без какого-либо значения, указанного для этого аргумента. Аргумент «…» является особенным и может содержать любое число аргументов. Он обычно используется, если число аргументов неизвестно или в случаях, где аргументы будут переданы другой функции.

Тело – синтаксически проанализированный оператор ***R,*** обычно набор операторов в фигурных скобках, но также может быть отдельный оператор, символ или даже константа.

Окружающая среда функции является средой, которая была активной при создании функции. Любой символ ограничен своей окружающей средой, связан и доступен функции. Комбинацию кода функции и привязки в ее окружающей среде называют «оберткой функции», термином из теории функционального программирования. Здесь обычно используется термин «функция», но используется «обертка», чтобы подчеркнуть значимость присоединенной среды.

Можно извлечь и управлять тремя частями обертки объекта, используя конструкции *formals, body* и *environment* (все три могут также использоваться на левой стороне присваивания). Последний из них может использоваться для удаления нежелательной привязки среды.

При вызове функции создается новая среда (называемая средой оценки), чье пространство (см. раздел 2.1.10 [Окружающая среда]) является средой от обертки функции. Новая среда первоначально заполнена неоцененными аргументами функции; поскольку оценка продолжается, локальные переменные создаются в ее пределах.

Есть также средство для преобразования функции «в» и «из» списочной структуры, используя *as.list* и *as. function*. Они были включены для совместимости с ***S*** и их использование обескураживает.

2.1.7. NULL

Существует специальный объект, называемый *NULL.* Он используется всякий раз, когда есть потребность идентифицировать или указать отсутствие объекта. Его не следует путать с вектором или списком нулевой длины.

Объект *NULL* не имеет типа и каких-либо поддающихся изменению свойств. В ***R*** есть только один объект *NULL*, к которому обращаются все экземпляры. Для проверки на *NULL* используют *is.null*. Нельзя установить атрибуты для *NULL.*

2.1.8. Встроенные объекты и специальные формы

Эти два вида объекта содержат встроенные функции ***R,*** то есть, те, которые выведены на экран как. *Primitive* в листингах кода (так же как те, к которым получают доступ через функцию. *Internal* и, следовательно, не видимые пользователем как объекты). Различия между ними заключается в обработке аргумента. Все собственные аргументы встроенных функций оцени-

ваются и передаются внутренней функции в соответствии с *вызовом по значению,* тогда как специальные функции передают не оцененные аргументы внутренней функции.

Для языка ***R*** эти объекты – только другой вид функции. Функция *is.primitive* может отличить их от интерпретируемых функций.

2.1.9. Обещанные объекты

Объекты обещания – часть механизма отложенных вычислений ***R.*** Они содержат три слота: значение, выражение и окружающая среда. При вызове функции сравниваются аргументы, а затем каждый из формальных аргументов является обязательством к обещанию. Выражение, которое было дано для формального аргумента, и указатель на окружающую среду функции вызываются из сохраненных в обещании.

Пока к этому аргументу не получают доступ, не существует какого-либо значения, присоединенного с обещанием. Когда к аргументу получают доступ, сохраненное выражение оценивается в сохраненной окружающей среде и возвращается результат. Результат также сохранен обещанием. Функция замены извлечет контент слота выражения, что позволяет программисту получать доступ или к значению или к выражению, присоединенному с обещанием.

В пределах языка ***R*** объекты обещания видимы почти неявно: фактические аргументы функции имеют этот тип. Есть также функция *delayedAssign*, которая сделает обещание из выражения. Отсутствует какой-либо способ в коде ***R*** для проверки, является ли объект обещанием или нет, и при этом нет способа использовать код ***R*** для определения окружающей среды обещания.

2.1.10. Точка-точка-точка

Тип объекта «…» хранится как тип *pairlist*. К компонентам «…» можно получить доступ обычным *pairlist* способом из кода ***C,*** но не легко получить доступ как к объекту в интерпретируемом коде. Объект может быть получен как список, так, например, в таблице ниже:

```
args <– list (…)
##…
for (a in args) {
##…
```

Если функция имеет «…» в качестве формального аргумента, который не соответствует формальным аргументам, то им сопоставляется «…».

2.1.11. Окружающая среда

Будем считать, что окружающие среды могут состоять из двух частей. *Фрейм*, состоящий из набора пар символ-значение, и *обертка* — указатель на обертку окружающей среды. Когда ***R*** ищет значение для символа, исследуется фрейм и, если соответствующий символ будет найден, его значение будет возвращено. В противном случае получают доступ к обертке окружающей среды, и процесс повторяется. Окружающие среды формируют древовидную структуру, в которой обертки играют роль родителей. Дерево окружающих сред имеет корень в пустой окружающей среде, доступной через *emptyenv (),* у которой нет родителя. Корень – прямой родитель окружающей среды основного *(base)* пакета (доступной через функцию *baseenv ()).* Прежде у *baseenv ()* было специальное значение NULL, но начиная с версии 2.4.0 использование NULL, является не допустимым в качестве окружающей среды.

Окружающие среды создаются неявно вызовами функции, как описано в разделе 2.1.5 [Объекты функции], и разделе 3.5.2 [Лексическая окружающая среда]. В этом случае окружающая среда содержит переменные, локальные для функции (включая аргументы), а ее обертка является окружающей средой вызванной в настоящий момент функции. Окружающие среды также можно создать непосредственно *new. env*. К контенту фрейма окружающей среды можно получить доступ и манипулирование при помощи *ls*, *get* и *assign*, включая *eval* и *evalq.*

Может использоваться функция *parent. env* для получения доступа к обертке окружающей среды.

В отличие от большинства других объектов ***R,*** окружающая среда не копируется при передаче в функцию или использовании в присвоениях. Таким образом, если присвоить одну и ту же окружающую среду нескольким символам и изменить одну из них, то другие изменятся также. В частности присвоение атрибутов окружающей среде может привести к неожиданностям.

2.1.12. Объекты парных списков

Объекты парных списков (Pairlist) подобны спискам точечной пары Lisp. Они интенсивно используются внутри ***R,*** но редко видимы в интерпретируемом коде, хотя они возвращаются *formals*, и могут быть созданы, например, функцией *pairlist*. *pairlist* с нулевой длиной равен NULL, как ожидался бы в Lisp, но в отличие от списка нулевой длины. У каждого такого объекта есть три слота, значение CAR, значение CDR и значение TAG. Значение TAG – текстовая строка, и CAR и CDR обычно представляют, соответственно, элемент списка (голова) и остаток (хвост) списка с объектом NULL как разделитель (терминология CAR/CDR – традиционна для Lisp и первоначально упоминалась как адрес и декрементные регистры на компьютерах IBM в начале 60-ых).

Парные списки обрабатываются на языке ***R*** точно таким же образом как универсальные векторы («*list*s»). В частности к элементам получают доступ, используя то же самый синтаксис [[]]. Использование парных списков является сомнительным, так как универсальные векторы обычно более эффективны в использовании. При получении доступа к внутреннему парному списку из ***R*** обычно (включая подмножества) преобразуют в универсальный вектор.

В очень немногих случаях парные списки видимы пользователем: одним из таких является. *Options*.

2.1.13. Тип «Any»

В действительности объект не может иметь тип «*Any*», что, однако, допустимо для значения типа. Случается при определенных (довольно редких) обстоятельствах, например, *as.vector (x,«any»)* указывает, что не следует делать приведение типа.

2.2. Атрибуты

У всех объектов кроме NULL могут быть один или более атрибутов, присоединенных к ним. Атрибуты сохранятся как *pairlist*, где все элементы именованы, но их следует рассматривать как ряд пар *name=value*. Можно получить список атрибутов, используя атрибуты и набор атрибутами <-, к отдельным компонентам получают доступ, используя *attr* и *attr* <-.

У некоторых атрибутов есть специальные функции доступа (например, *levels* <- для факторов), и они должны использоваться при доступности. В дополнение к скрытым деталям реализации они могут выполнить дополнительные операции. ***R*** пытается принять вызовы *attr* <- и *attributes* <-, которые включают специальные атрибуты, и осуществляет проверку непротиворечивости.

Матрицы и массивы – просто векторы с атрибутом размерности и дополнительно с именем размерности (*dimnames*), присоединенные к вектору.

Атрибуты используются для реализации структуры класса, используемой в ***R.*** Если у объекта есть атрибут класса, то атрибут будет исследован во время оценки. Структура класса в ***R*** описана подробно в главе 5 [Объектно-ориентированное программирование].

2.2.1. Имена

Атрибут имени (*names*) при его присутствии маркирует отдельные элементы вектора или списка. При печати объекта атрибут имени при его присутствии используется для маркировки элементов. Атрибут имен может также использоваться для индексирования результата, например, *quantile (x) [«25%»].*

Можно получить и определить имена, используя конструкции *names* и *names* <-. Последняя выполнит необходимые проверки непротиворечивости, чтобы гарантировать, что у атрибута имен есть надлежащий тип и длина.

Отдельно обрабатываются парные списки и одномерные массивы. Для объектов парных списков (*pairlist*) используется виртуальный атрибут имен; в действительности атрибут имен создается из тегов компонентов списка. Для одномерных массивов реально имена доступы из имен размерности (*dimnames [[1]]).*

2.2.2. Размерность

Атрибут размерности используется при реализации массивов. Контент массива сохраняется в векторе в порядке по столбцам, и атрибут размерности является вектором целых чисел, указывающие соответствующие пределы массива. ***R*** гарантирует, что длина вектора равна произведению длин размерностей. Длина одной или более размерностей может равняться нулю.

Вектор не является одномерным массивом, так как у последнего атрибут размерности длиною один, тогда как у вектора атрибут размерности отсутствует.

2.2.3. Имена размерности

Массивы могут назвать каждую размерность, отдельно используя атрибут имен размерности (*dimnames*), который является списком символьных векторов. У списка имен размерностей самостоятельно могут быть имена, которые затем используются для заголовков пределов при печати массива.

2.2.4. Классы

У ***R*** есть тщательно продуманная система классов, которая преимущественно управляется через атрибут класса. Этот атрибут – символьный вектор, содержащий список классов, от которых наследовался объект. Он формирует основание «универсальных методов» функциональности ***R.***

К этому атрибуту пользователи могут получить доступ и управление фактически без ограничения. Отсутствует проверка, что объект фактически содержит компоненты, которые ожидают методы класса. Таким образом, изменение атрибута класса должно быть сделано

с осторожностью, и при доступе к ним следует предпочесть определенные функции создания и преобразования.

2.2.5. Атрибуты временных рядов

Атрибут *tsp* используется для поддержания аргументов временного ряда: начало, конец и частота. Эта конструкция, главным образом, используется для обработки рядов с периодической подструктурой, например, месячные или квартальные данные.

2.2.6. Копирование атрибутов

Атрибуты копируются при изменении объекта, находящегося в комплексной области, по некоторым общим правилам (Becker, Chambers & Wilks, 1988, стр. 144—6).

Скалярные функции (те, которые работают поэлементно на векторе и чей выход подобен входу) должны сохранить атрибуты (кроме, возможно, класса).

Бинарные операции обычно копируют большинство атрибутов более длинного аргумента (и если они имеют одинаковую длину для обоих, то предпочитается значение для первого). Здесь «большинство» означает все кроме имен, размерностей и имен размерностей, которые установлены соответственно кодом для оператора.

Подмножество (кроме индексированных пустым индексом) обычно отбрасывает все атрибуты кроме имен, размерностей и имен размерностей, которые сброшены как соответствующие. С другой стороны подприсвоение обычно сохраняет атрибуты, даже если длина изменена. Приведение отбрасывает все атрибуты.

Метод по умолчанию для сортировки отбрасывает все атрибуты кроме имен, которые сортируются наряду с объектом.

2.3. Специальные составные объекты

2.3.1. Факторы

Факторы используются для описания элементов, у которых может быть конечное количество значений (пол, социальный класс, и т.д.). У фактора есть атрибут уровней и класс «фактор» *(factor*). Дополнительно, он также может содержать атрибут *contrasts*, который управляет параметризацией при использовании факторов в функциях моделирования.

Фактор может быть просто номинальным или, возможно, иметь упорядоченные категории. В последнем случае он должен быть определен как таковой и иметь вектор класса (*class) c («ordered»,» factor»)*.

В настоящий момент факторы реализованы с использованием целочисленного массива для указания фактических уровней, и второй массив имен, которые отображены на целые числа. Скорее, к сожалению, пользователи часто используют реализацию для облегчения некоторых вычислений. Это, однако, является проблемой реализации и, как гарантируют, не будет иметь место во всех реализациях ***R.***

2.3.2. Объект фрейм данных

Фреймы данных является структурой ***R,*** которая наиболее близко подражает SAS или набору данных SPSS, то есть «переключение переменными ” матриц данных.

Фрейм данных является списком векторов, факторов, и/или матриц, имеющих одинаковую длину (число строк в случае матриц). Кроме того, у фрейма данных обычно есть атрибут имен, маркирующий переменные и атрибут *row.names* для маркировки переключателей *(cases*).

Фрейм данных может содержать список, который имеет одинаковую длину с другими компонентами. Список может содержать элементы разных длин, таким образом, обеспечивая структуру данных для массивов с переменной длиной строк. Однако для таких записей обычно массивы обрабатываются не правильно.

3. Оценка выражений

При вводе команды пользователем в запросе (или при считывании выражения из файла), сначала команды преобразуются синтаксическим анализатором во внутреннее представление. Средство оценки выполняет синтаксический анализ выражения ***R*** и возвращает значение выражения. У всех выражений есть значение. Это – основа языка.

Данная глава описывает основные механизмы средств оценки, но избегает обсуждения определенных функций или групп функций, которые описаны позже в отдельных главах или где страницы справки должны быть достаточной документацией.

Пользователи могут создать выражения и вызвать их оценку.

3.1. Простая оценка

3.1.1. Константы

Любое число, введенное непосредственно в запросе, является константой и оценивается.

```
> 1
[1] 1
```

Возможно, неожиданно, но число, возвращенное из выражения 1, является действительным. В большинстве случаев разница между целым числом и действительным значением будет незначительна, поскольку ***R*** делает округление справа при использовании чисел. Однако может возникнуть необходимость создания целочисленного значения для константы, что можно сделать, вызывая функцию *as.integer* или используя различные другие методы. Но возможно самый простой подход состоит в сопровождении константы символьным суффиксом «*L*». Например, для создания целочисленного значения 1 можно использовать:

```
> 1L
[1]
```

Можно использовать суффикс *«L»*, чтобы квалифицировать любое число с намерением создания из него явно целое число. Таким образом, «*0x10L*» создает целочисленное значение *16* из шестнадцатеричного представления. Константа *1e3L* дает *1000* как целое число, а не числовое значение и эквивалентна *1000L*. Заметим, что *«L»* обработан как квалификация аргумента *1e3* а не *3*. Если квалифицируем значение с *«L»*, которое не является целочисленным значением, например, *1e-3L,* то получаем предупреждение, и создается действительное значение. Также получаем предупреждение при наличии ненужной десятичной точки в числе, например, *1.L*.

Получим синтаксическую ошибку при использовании «***L***» с комплексными числами, например, *12iL* дает ошибку.

Константы являются довольно скучными, и не будем больше тратить слова.

3.1.2. Просмотр символов

При создании новой переменной у нее должно быть имя, что дает возможность на нее сослаться, и у нее обычно есть значение. Само имя – символ. При оценке символа возвращается его значение. Позже объясним подробно, как определить значение, которое имеет символ.

В этом небольшом примере ***y*** – символ и его значение ***4***. Символ также является объектом ***R,*** но редко возникает необходимость иметь дело с символами непосредственно, кроме случаев «Программирование на языке» (Глава 6 [Вычисления на языке], страница 32).

```
> y <– 4
> y
[1] 4
```

3.1.3. Вызов функции

Большинство вычислений, выполненных в ***R,*** включает оценку функций, называемых как *вызов* функции. Функции вызываются по имени со списком аргументов, разделенных запятыми.

```
> mean (1:10)
```

[1] 5.5

В этом примере функция *mean* (средняя) была вызвана с одним аргументом, вектором целых чисел от 1 до 10.

R содержит огромное число функций с различными результатами. Большинство используется для получения результата, который является объектом ***R,*** но некоторые используются для вспомогательных целей, например, функции печати и рисования.

Вызовы функции могут тегировать (или называть) аргументы, как в *plot (x, y, pch = 3),* аргументы без тегов известны как позиционные, так как функция должна отличить их значение от их последовательных позиций среди аргументов вызова, например, что ***x*** обозначает переменную абсциссы, а ***y*** ординату. Использование тегов/имен – очевидное удобство для функций с большим количеством дополнительных аргументов.

Специальный тип вызовов функции может появиться на левой стороне оператора присваивания как в:

> class (x) <– «foo»

В действительности вызывается функция *class <–* с исходным объектом и правой стороной. Функция выполняет модификацию объекта и возвращает результат, который затем сохраняется обратно в исходной переменной. По крайней мере, концептуально так должно быть. Прилагаются дополнительные усилия для исключения ненужного дублирования данных.

3.1.4. Операторы

R позволяет использование арифметических выражений с помощью операторов, подобных таковым из языка программирования ***C,*** например:

> 1 +2
[1] 3

Используя круглые скобки, выражения можно сгруппировать с включением вызовов функций, и прямым присвоением переменным:

*> y <– 2 * (a + log (x))*

R содержит много операторов. Они перечислены в таблице ниже.

–	*Минус, может быть унарным или бинарным*
+	*Плюс, может быть унарным или бинарным*
!	*Унарное нет*

~

Тильда, используемая для формул модели, может быть или унарным или бинарным

?

Справка

:

Последовательность, двоичная (в формулах модели: взаимодействие)

*

Умножение бинарное

/

Деление бинарное

^

Возведение в степень бинарное

%x%

Специальные бинарные операторы, x могут быть заменены любым допустимым именем

%%

Модуль бинарный

% / %

Целочисленное деление, бинарное

*% * %*

Матричное произведение, бинарное

%o%

Внешнее произведение, бинарное

%x%

Кронекерово умножение, бинарное

%in%

Соответствие оператора, бинарного (в формулах модели: гнездованое)

<

Меньше чем, бинарный

>

Больше чем, бинарный

==

Равно, бинарное

> =

Больше чем или равно, бинарное

<=

Меньше чем или равно, бинарное

& And,

И бинарное, векторизовано

&&	*И бинарное, не векторизовано*
|	*Или бинарное, векторизовано*
||	*Или бинарное, не векторизовано*
<-	*Левое присвоение, бинарное*
– >	*Правое присвоение, бинарное*
$	*Подмножество списка, бинарное*

За исключением синтаксиса, нет никакой разницы между применением оператора и вызовом функции. Фактически, *x + y* может эквивалентно быть записано *‘+‘ (x, y).* Заметим, что так как ‘+’ не является именем стандартной функции, то он должен быть заключен в кавычки.

R имеет дело со всем вектором данных за один раз, и большинство элементарных операторов и основных математических функций, например, *log* являются векторизованными (как обозначено в таблице выше). Например, добавление двух векторов одинаковой длины создаст вектор, содержащий поэлементные суммы, неявно индексируя циклическое выполнение по вектору. Также применяют как к другим операторам, таким ка – *, и / так и к структурам более высокой размерности. Заметим в особенности, что умножение двух матриц не производит обычное матричное произведение (оператор ***%*%*** существует с этой целью). Некоторые тонкости, касающиеся векторизованных операций, будут обсуждены в разделе 3.3 [Элементарные арифметические операции].

Для получения доступа к отдельным атомарным элементам вектора используется конструкция *x [i]:*

```
> x <– rnorm (5)
> x
[1]
```

– 0.12526937 -0.27961154 -1.03718717 -0.08156527 1.37167090

```
> x [2]
[1] -0.2796115
```

Для доступа к компоненте списка обычно используется *x$a* или *x [[i]]*.

```
> x <– options ()
> x$prompt
[1] ">»
```

Также можно использовать индексацию конструкций на правой стороне присвоения.

Подобно другим операторам, индексация, в действительности, выполняется функциями, и можно использовать *« [» (x, 2)* вместо *x [2]*.

Операции индексации ***R*** содержат много расширенных функций, которые описаны далее в разделе 3.4 (Индексирование).

3.2. Управляющие структуры

Вычисление в ***R*** состоит в последовательной оценке *операторов*. Операторы, такие как *x <-1:10* или *mean (y)*, могут быть разделены или точкой с запятой или новой строкой. Всякий раз, когда средству анализа предоставляют синтаксически полный оператор, этот оператор оценивается и возвращается значение. Результат оценки оператора может упоминаться как значение оператора. Значение всегда присваиваться символу.

Для разделения операторов могут использоваться и точки с запятой, и новые строки. Точка с запятой всегда указывает на конец оператора, в то время как новая строка может указывать на конец оператора. Если текущий оператор синтаксически не полный, то новые строки просто игнорируются средством анализа. Если сеанс является интерактивным, запрос изменяется с **‘> ’** на **‘+».**

```
> x <– 0; x +5 [1] 5
> y <– 1:10
> 1; 2
[1] 1
[1] 2
```

Операторы могут группироваться использованием фигурных скобок ***‘ {»*** и***»}»***. Группу операторов иногда вызывают блоком. Отдельные операторы оцениваются при вводе новой строки в конце синтаксически полного оператора. Блоки не оцениваются, пока новая строка не вводится после закрывающей фигурной скобки. В оставшейся части этого раздела оператор ссылается на отдельный оператор или блок.

```
> {x <– 0
+ x +5
+}
[1] 5
```

3.2.1. Оператор *if*

Оператор *if/else* условно оценивает два оператора. Существует условие, которое подлежит оценке, и если значение равно TRUE, то первый оператор оценивается; иначе оценивается второй оператор. Оператор *if/else* возвращает в качестве своего значения значение выбранного оператора. Формальный синтаксис таков:

```
if (statement1)
statement2
else
statement3
```

Во-первых, оценивается *statement1* для получения *value1*. Если *value1* – логический вектор с первым элементом, равным TRUE, то оценивается *statement2*. Если первый элемент *value1* равен FALSE, то оценивается *statement3*. Если *value1* – числовой вектор, то оценивается *statement3*, когда первый элемент *value1* равен нулю, а иначе оценивается *statement2*. Используется только первый элемент *value1*. Все другие элементы игнорируются. Если у *value1* есть какой-либо тип кроме логического или числового вектора, то сигнализируется ошибка.

Можно использовать оператор *if/else* для исключения числовых проблем, таких как взятие логарифма отрицательного числа. Поскольку, оператор *if/else*

Конец ознакомительного фрагмента.