2
Синтаксис выражений

Литералы

Числа

Символы

Строки

Имена

Массивы

Переменные

Присваивания

Имена псевдопеременных

Сообщения

Имена сообщений и аргументы

Возвращение значений

Синтаксический анализ

Соглашения о форматировании

Каскадирование

Блоки

Управляющие структуры

Условные выражения

Аргументы блока

Словарь терминов

В первой главе были введены основные понятия системы Smalltalk-80. Элементы системы представляются *объектами*. Объекты — *экземпляры классов*. Объекты взаимодействуют между собой, посылая друг другу *сообщения*. Сообщения приводят к выполнению *методов*. Эта глава знакомит с синтаксисом выражений, описывающих объекты и сообщения. Следующая глава знакомит с синтаксисом описания классов и методов.

Выражение — последовательность символов, описывающая объект, который называется значением выражения. Синтаксис, представленный в главе, объясняет, какие последовательности символов составляют правильные выражения. В языке программирования Smalltalk-80 существуют четыре типа выражений.

1. *Литералы* описывают некоторые постоянные объекты, такие как числа или строки символов.
2. *Имена переменных* описывают доступные переменные. Значение имени переменной — текущее значение переменной с этим именем.
3. *Выражения посылки сообщения* описывают сообщения, посланные получателю. Значение такого выражения определяется тем методом, которое вызывается сообщением. Метод ищется в классе получателя.
4. *Выражения-блоки* описывают объекты, представляющие отложенные действия. Блоки применяются для реализации управляющих структур.

Выражения находятся в двух местах: в методах и в тексте, изображенном на экране. Когда посылается сообщение, из класса получателя выбирается метод и его выражения выполняются. Интерфейс пользователя позволяет выбирать и выполнять выражения на экране. Подробности этого процесса выходят за пределы данной книги, так как являются частью интерфейса пользователя. Тем не менее, некоторые примеры выполнения выражений на экране приводятся в главе 17.

Из четырех типов перечисленных выше выражений только имена переменных зависят от контекста. Местонахождение выражения в системе определяет, какие последовательности символов будут правильными именами переменных. Множество имен переменных, допустимых в выражениях методов, зависят от класса, в котором находится метод. Например, методы в классе Rectangle и методы в классе Point имеют доступ к различным множествам имен переменных. Переменные, доступные в методах класса, будут полностью описаны в главах 3, 4 и 5. Имена переменных, доступные для использования в выражениях на экране, зависят от того, в каком месте экрана выражения отображаются. В остальном синтаксис выражения не зависит от местоположения последнего.

Синтаксис выражений схематично представлен на диаграмме в конце книги. Оставшаяся часть главы описывает приведенные выше четыре типа выражений.

Литералы

На пять видов объектов можно ссылаться с помощью литеральных выражений. Так как значение литерального выражения всегда один и тот же объект, такое выражение называют *литеральной константой*. Есть пять типов литеральных констант:

1. числа
2. отдельные символы
3. строки символов
4. имена
5. массивы других литеральных констант

Числа

Числа — объекты системы, которые представляют числовые значения и отвечают на сообщения, вычисляющие математические результаты. Литеральное представление числа со-

стоит из последовательности цифр, которой может предшествовать знак минус, или/и за которой может следовать десятичная точка и другая последовательность цифр. Например:

```
3
30.45
–3
–14.0
13772
```

Числовые константы также могут быть выражены и не в десятичной системе счисления с помощью предшествующего цифрам префикса основания системы счисления. Префикс состоит из цифровой величины основания (всегда выраженной в десятичной системе) и следующей за ней буквы “r”. Следующие примеры представляют числа в восьмеричной системе и соответствующие им десятичные значения.

восьмеричная форма	*десятичная форма*
8r377	255
8r153	107
8r34.1	28.125
8r–37	–31

Когда основание системы счисления больше десяти, цифры большие девяти обозначаются прописными латинскими буквами, начиная с “A”. Следующие примеры представляют числа в шестнадцатеричной системе и соответствующие им десятичные значения.

Шестнадцатеричная форма	*десятичная форма*
16r106	262
16rFF	255
16rAC.DC	172.859
16r–1.C	–1.75

Числовые константы также могут быть выражены в экспоненциальной форме с помощью экспоненциального суффикса. Экспоненциальный суффикс состоит из (латинской) буквы “e” и следующего за ним порядка, выраженного в десятичной системе счисления. Число, определенное перед экспоненциальным суффиксом, умножается на основание системы счисления, возведенное в степень, заданную порядком.

Экспоненциальная форма	*десятичная форма*
1.586e5	158600.0
1.586e-3	0.001586
8r3e2	192
2r11e6	192

Символы

Символы — объекты, которые представляют собой отдельные элементы алфавита. Символьное литеральное выражение состоит из символа доллара “$” и следующего за ним произвольного символа алфавита[1]. Например:

```
$t
$Ф
$–
```

[1] Помимо стандартных символов латинского алфавита, в символьных константах, а также в строках и комментариях допускаются символы национальных алфавитов. — Прим. ред.

```
$$
$1
```

Строки

Строки — это объекты, которые представляют собой последовательности символов. Строки отвечают на сообщения, которые осуществляют доступ к отдельным символам, заменяют подстроки и выполняют сравнения с другими строками. Литеральное представление строки — это последовательность символов, заключенная в апострофы, например:

```
'привет'
'food'
'Система Smalltalk-80'
```

Любой символ может быть включен в строку. Если апостроф сам входит в строку, то он должен дублироваться, чтобы избежать путаницы с ограничителем. Например, строковая литеральная константа

```
'can''t'
```

представляет строку из пяти символов $c, $a, $n, $', $t.

Имена

Имена — это объекты, которые представляют собой строки для именования элементов системы. Литеральное представление имени — последовательность символов, начинающаяся символом “#”, например:

```
#bill
#M63
```

Есть некоторые ограничения на символы, которые могут быть в именах. Синтаксические диаграммы в конце книги дают точные определения. Не существует разных имен с одинаковым набором символов; в системе каждое имя уникально. Это позволяет эффективно сравнивать имена.

Массивы

Массив — простая структура данных, на элементы которой можно ссылаться целыми индексами от 1 до числа, равного размеру массива. Массив отвечает на сообщения доступа к его элементам. Литеральное представление массива — символ “#”, за которым следует заключенная в круглые скобки последовательность других литералов — чисел, символов, строк, имен и массивов. Элементы массива отделяются пробелами. Перед вложенными в массив именами и массивами не ставится символ “#”. Например, массив из трех чисел описывается выражением

```
#(1 2 3)
```

Массив из семи строк описывается выражением

```
#('food' 'utilities' 'rent' 'household' 'transport' 'taxes' 'recreation')
```

Массив из двух массивов и двух чисел описывается выражением

```
#(('one' 1) ('not' 'negative') 0 –1)
```

Массив, состоящий из числа, строки, символа, имени и другого массива описывается выражением

```
#(9 'nine' $9 nine (0 'zero' $0 () 'e' $f 'g' $h 'i'))
```

Переменные

Доступная объекту память состоит из переменных. Большинство таких переменных имеют имена. Каждая переменная запоминает один объект и имя переменной может быть использовано как выражение, ссылающееся на этот объект. Объекты, к которым можно получить доступ из конкретного места программы, определяются через доступные имена переменных. Например, содержимое переменных экземпляра некоторого объекта не доступно для других объектов, потому что имена этих переменных могут быть использованы только в методах того класса, которому принадлежит объект.

Имя переменной — простой идентификатор, то есть последовательность букв и цифр, начинающаяся с буквы. Например,

```
index          bin14Total
initialIndex   HouseholdFinances
textEditor     Rectangle
bin14          IncomeReasons
```

Существует два вида переменных в системе, отличающиеся тем, насколько широко они доступны. Частные переменные доступны только для одного объекта. Переменные экземпляра — частные. Общие переменные могут быть доступны для более чем одного объекта. Имена частных переменных должны начинаться со строчной буквы, а общих переменных — с прописной. Первые пять примеров идентификаторов, приведенные выше, относятся к частным переменным, а три последних — к общим.

Другое соглашение о написании, ясное из приведенного примера, состоит в том, что составные имена образуются слитным написанием нескольких слов, где каждое новое слово начинается с прописной буквы. Это соглашение, однако, не контролируется системой.

Присваивания

Литеральные константы всегда ссылаются на один и тот же объект, а имена переменных в разное время могут ссылаться на различные объекты. Ссылка на объект по имени переменной изменяется тогда, когда выполняется выражение присваивания. Присваивание не выделяется как отдельный тип выражения, поскольку любое выражение может стать присваиванием после включения в него префикса присваивания.

Префикс присваивания состоит из имени переменной, значение которой будет изменено, за которым следует левая стрелка "←"[2]. Следующий пример — литеральное выражение с префиксом присваивания. Оно указывает, что переменная с именем quantity будет теперь ссылаться на объект, представляющий число 19:

```
quantity ← 19
```

Следующий пример — выражение-переменная с префиксом присваивания. Оно указывает, что переменная с именем index будет далее ссылаться на тот же объект, что и переменная с именем initialIndex:

```
index ← initialIndex
```

Вот еще примеры выражений присваивания:

```
chapterName ← ExpressionSyntax
flavor ← #('vanilin' 'chocolate' 'butter pecan' 'garlic')
```

Можно ставить и более одного префикса присваивания, указывая на то, что изменяются значения нескольких переменных. Например,

```
index ← initialIndex ← 1
```

[2] В последних версиях языка и различных системах Smalltalk левую стрелку можно заменять на более привычную пару символов ":=". — Прим. ред.

Последний пример показывает, что переменные с именами index и initialIndex должны ссылаться на число 1. Выражения посылки сообщения и выражения-блоки также могут иметь префикс присваивания, что будет показано в следующих разделах.

Имена псевдопеременных

Имя псевдопеременной — идентификатор, который ссылается на объект. Этим оно похоже на имя переменной. Однако значение имени псевдопеременной, в отличие от имени переменной, не может быть изменено с помощью выражения присваивания. Некоторые псевдопеременные в системе — константы; они всегда ссылаются на одни и те же объекты. Три таких важных псевдопеременных — nil, true и false.

nil (нуль)	ссылается на объект, используемый как значение переменной, которой не соответствует никакой объект. Переменные, которые не были инициализированы, ссылаются на nil.
true (истина)	ссылается на объект, который представляет логическую истину. Используется как утвердительный ответ на сообщение, задающее простой вопрос, который требует ответа “да” или “нет”.
false (ложь)	ссылается на объект, который представляет логическую ложь. Используется как отрицательный ответ на сообщение, задающее простой вопрос, который требует ответа “да” или “нет”.

Объекты с именами true и false называются логическими (булевыми) объектами. Они представляют собой ответы на вопрос “да-нет”. Например, true или false будет ответом на сообщение, спрашивающее число о том, больше ли оно другого числа. Логические объекты отвечают на сообщения, которые вычисляют логические функции или выполняют условные управляющие структуры. В системе есть и другие псевдопеременные, например, self (сам) и super (супер), чьи значения зависят от того, где эти псевдопеременные используются. Они будут описаны в следующих трех главах.

Сообщения

Сообщения представляют взаимодействия компонентов системы Smalltalk-80. Сообщение запрашивает операцию над частью получателя сообщения. Некоторые примеры выражений посылки сообщений и представляемых ими взаимодействий объектов приведены ниже.

Сообщения числам, приводящие к выполнению арифметических операций:

3 + 4	вычисляет сумму трех и четырех.
index + 1	добавляет единицу к числу с именем index.
index > limit	спрашивает, больше ли число с именем index, чем число с именем limit.
theta sin	вычисляет синус числа с именем theta.
quantity sqrt	вычисляет положительный квадратный корень из числа с именем quantity.

Сообщения линейным структурам данных, представляющие добавление или удаление информации:

list addFirst: newComponent	добавляет объект с именем newComponent как первый компонент линейной структуры данных с именем list.
list removeLast	удаляет последний элемент из структуры list.

Сообщения ассоциативным структурам данных (таким как словари), представляющие добавление и удаление информации:

ages at: 'Brett Jorgensen' put: 3

ставит в соответствие строке 'Brett Jorgensen' число 3 в словаре с именем ages.

addresses at: 'Peggy Jorgensen'

ищет объект, связанный со строкой 'Peggy Jorgensen' в словаре с именем addresses.

Сообщения прямоугольникам, представляющие графические запросы и вычисления:

frame center — выдает позицию центра прямоугольника с именем frame.

frame containsPoint: cursorLocation

выдает значение true, если позиция с именем cursorLocation находится внутри прямоугольника с именем frame, и значение false в противном случае.

frame intersect: clippingBox

выдает прямоугольник, который представляет собой пересечение двух прямоугольников с именами frame и clippingBox.

Сообщения экземплярам класса FinancialHistory, представляющие описания сделок и запросы:

HouseholdFinances spend: 32.50 for: 'utilities'

информирует HouseholdFinances, что 32.50 доллара было потрачено на оплату счета за коммунальные услуги.

HouseholdFinances totalSpentFor: 'food'

запрашивает HouseholdFinances, сколько денег было потрачено на питание.

Имена сообщений и аргументы

Выражение посылки сообщения состоит из получателя, *имени сообщения*, и, возможно, нескольких *аргументов*. Получатель и аргументы описываются другими выражениями. Имя сообщения задается литералом.

Имя сообщения представляет собой название типа взаимодействия, которое отправитель сообщения желает выполнить с получателем. Например, в сообщении

```
theta sin
```

получатель — число, на которое ссылается переменная с именем theta, а имя сообщения — sin. Получатель сам решает, как ответить на сообщение, в данном случае, как вычислить функцию синуса от своего значения.

В следующих двух сообщениях

```
3 + 4
```

и

```
previousTotal + increment
```

имя сообщения — символ “+”. Оба сообщения просят получателя вычислить сумму. Каждое из сообщений содержит объект в дополнение к имени сообщения (4 в первом выражении и increment во втором). Дополнительные объекты в сообщении — это аргументы, определяющие добавляемое значение.

Два следующих выражения посылки сообщения описывают одинаковые виды операций. Получатель, экземпляр класса FinancialHistory, выдаст сумму денег, потраченную на некоторый вид расходов. Аргумент задает вид расходов. Первое выражение запрашивает сумму, потраченную на оплату счетов за коммунальные услуги:

```
HouseholdFinances totalSpentFor: 'utilities'
```

Сумма, потраченная на продукты питания, может быть найдена с помощью сообщения с тем же самым именем, но другим аргументом:

```
HouseholdFinances totalSpentFor: 'food'
```

Имя сообщения определяет, какая из операций получателя будет вызвана. Аргументы представляют другие объекты, которые вовлекаются в выбранную операцию.

☐ *Унарные сообщения* Сообщения без аргументов называются *унарными сообщениями*. Например, текущая сумма денег на счете, представленном объектом HouseholdFinances, является значением выражения, представленного унарным сообщением

```
HouseholdFinances cashOnHand
```

Такие сообщения названы унарными потому, что вовлекают в операцию только один объект — получатель сообщения. Имя унарного сообщения может быть любым простым идентификатором. Вот другие примеры унарных сообщений:

```
theta sin
quantity sqrt
namestring size
```

☐ *Ключевые сообщения* Общий тип сообщения с одним или более аргументами — это *ключевое сообщение*. Имя ключевого сообщения составляется из одного или более ключевых слов, по одному перед каждым аргументом. Ключевое слово — идентификатор с последующим двоеточием. Примеры ключевых сообщений:

```
HouseholdFinances totalSpentFor: 'utilities'
index max: limit
```

Сообщение с двумя аргументами будет иметь имя с двумя ключевыми словами. Примеры сообщений с двумя ключевыми словами:

```
HouseholdFinances spend: 30.45 for: 'food'
ages at: 'Brett Jorgensen' put: 3
```

Когда на имя многоаргументного ключевого сообщения ссылаются независимо от получателя этого сообщения, ключевые слова сливаются воедино. Так, например, имена двух последних сообщений — spend:for: и at:put:. Число ключевых слов в сообщении может быть любым, но большинство сообщений в системе имеют их не более трех.

☐ *Бинарные сообщения* Существует другой тип выражений посылки сообщений с одним аргументом. Это *бинарные сообщения*. Имя бинарного сообщения состоит из одного или двух не алфавитно-цифровых символов. Единственное ограничение здесь то, что второй символ не может быть знаком минус. Бинарные сообщения обычно используются для арифметических операций. Например,

```
3 + 4
total - 1
total <= max
```

Возвращение значений

Сообщения Smalltalk-80 обеспечивают двустороннюю связь. Имя и аргументы сообщения передают получателю информацию о том, какой тип ответа нужно подготовить. Получатель передает информацию обратно, возвращая отправителю объект, который становится значением выражения посылки сообщения. Если выражение включает префикс присваивания, то объект, возвращенный получателем, станет новым объектом, на который будет ссылаться переменная. Например, выражение

```
sum ← 3 + 4
```

устанавливает, что число 7 будет новым значением переменной с именем sum. Выражение

```
x ← theta sin
```

устанавливает синус числа theta как новое значение переменной с именем x. Если значение theta равно 1, то новое значение x становиться равным 0.841471; если же значение theta равно 1.5, то новое значение x становиться равным 0.997495.

Число, на которое ссылается переменная index, может быть увеличено с помощью выражения

```
index ← index + 1
```

Даже если нет никакой информации, которую нужно передать обратно отправителю, получатель всегда возвращает некоторое значение. Возвращение значения показывает, что обработка получателем полученного сообщения завершена. Некоторые сообщения служат только чтобы информировать получателя о чем-либо. Примеры таких сообщений для записи финансовых операций, описаны следующими выражениями:

```
HouseholdFinances spend: 32.50 for: 'utilities'
HouseholdFinances receive: 1000 from: 'pay'
```

Получатель сообщения информирует отправителя только об окончании записи сделки. Значением по умолчанию, возвращаемым в таких случаях, обычно является сам получатель. Так, выражение

```
var ← HouseholdFinances spend: 32.50 for: 'utilities'
```

приводит к тому, что переменная var будет ссылаться на тот же финансовый отчет, что и HouseholdFinances.

Синтаксический анализ

До сих пор все приводимые выше выражения посылки сообщения описывали получателя и аргументы литеральными константами или именами переменных. Когда же получатель или аргумент выражения посылки сообщения описываются другим выражением посылки сообщения, встает вопрос: как анализируется и в каком порядке выполняется такое выражение? Вот пример унарного сообщения, получателем которого является другое унарное сообщение:

```
1.5 tan rounded
```

Унарные сообщения посылаются в порядке слева направо. Первое сообщение в примере — унарное с именем tan — посылается числу 1.5. Значение этого выражения (число приблизительно равное 14.1014) затем получает унарное сообщение rounded и возвращает ближайшее целое 14. Число 14 становится значением всего выражения.

Очередь посылки бинарных сообщений также слева направо. Вот пример, в котором одно бинарное сообщение описывает получателя другого бинарного сообщения:

```
index + offset * 2
```

Значение, возвращаемое числом index в ответ на бинарное сообщение + offset, становится получателем для бинарного сообщения * 2.

В языке Smalltalk-80 все бинарные сообщения имеют одинаковый приоритет и для выполнения выражения имеет значение только порядок, в котором они записаны. Заметьте, что это отличает математические выражения на языке Smalltalk-80 от других языков программирования, в которых умножение и деление имеют более высокий приоритет, чем сложение и вычитание.

Для изменения порядка посылок сообщений можно применять круглые скобки. Сообщения, заключенные в скобки, посылаются раньше, чем сообщения за скобками. Если бы предыдущий пример был записан как

```
index + (offset * 2)
```

то умножение выполнилось бы раньше сложения.

Унарные сообщения имеют приоритет над бинарными. Если унарные и бинарные сообщения оказываются вместе в одном выражении, то сначала посылаются унарные сообщения. В примере

```
frame width + border width * 2
```

значение выражения frame width — это получатель бинарного сообщения с именем "+" и аргументом — значением выражения border width. В свою очередь значение сообщения "+" — это получатель бинарного сообщения * 2. Все это выражение разбирается так, как если бы в нем скобки были расставлены следующим образом:

```
((frame width) + (border width)) * 2
```

Скобки можно ставить для посылок бинарных сообщений перед унарными. Например, выражение

```
2 * theta sin
```

возвращает удвоенный синус числа theta, в то время как выражение

```
(2 * theta) sin
```

возвращает синус удвоенного числа theta.

Всякий раз, когда ключевые слова оказываются в сообщении без скобок, они составляют одно ключевое имя. Поэтому для ключевых сообщений не существует правила анализа “слева направо”. Если ключевое сообщение должно выступать как получатель или как аргумент другого сообщения, оно должно обязательно быть взято в скобки. Выражение

```
frame scale: factor max: 5
```

описывает единственное двухаргументное ключевое сообщение с именем scale:max:. Выражение

```
frame scale: (factor max: 5)
```

описывает два ключевых сообщения с именами scale: и max:. Значение выражения factor max: 5 становится аргументом для сообщения scale: объекту frame.

Бинарные сообщения имеют приоритет над ключевыми. Когда унарные, бинарные и ключевые сообщения оказываются в одном выражении без скобок, то сначала посылаются унарные, потом бинарные и последними ключевые сообщения. Выражение

```
bigFrame width: smallFrame width * 2
```

выполняется так, как если бы скобки были расставлены следующим образом:

```
bigFrame width: ((smallFrame width) * 2)
```

В следующем примере унарное сообщение описывает получателя ключевого сообщения, аргумент которого есть бинарное сообщение.

```
OrderedCollection new add: value * rate
```

Кратко приведем правила синтаксического разбора:

1. Унарные выражения разбираются слева направо.
2. Бинарные выражения разбираются слева направо.
3. Бинарные выражения имеют приоритет над ключевыми выражениями.
4. Унарные выражения имеют приоритет над бинарными выражениями.
5. Выражения, взятые в скобки, имеют приоритет над унарными выражениями.

Соглашения о форматировании

Программист свободен в выборе способов форматирования выражений и может по своему усмотрению вставлять в текст символы пробела, табуляции и возврата каретки. Например, в ключевом сообщении с несколькими словами часто каждую пару (ключевое слово, аргумент) записывают на разных строках, как в

```
ages at: 'Brett Jorgensen'
        put: 3
```

или в

```
HouseholdFinances
        spend: 30.45
        for: 'food'
```

Символы пробела, табуляции и возврата каретки влияют на смысл выражения только в том случае, когда их отсутствие приводит к слиянию рядом стоящих имен, слов, символов или чисел.

Каскадирование

Существует особая синтаксическая форма, называемая каскадированием, которая описывает посылку нескольких сообщений одному объекту. Любая последовательность сообщений может быть выражена и без использования этой формы. Однако каскадирование часто уменьшает необходимость в дополнительных переменных. Каскадное сообщение состоит из объекта-получателя, за которым располагаются несколько сообщений, разделенных точкой с запятой. Например:

```
OrderedCollection new add: 1; add: 2; add: 3
```

Три сообщения add: посылаются результату выражения OrderedCollection new. Без каскадирования для получения того же самого результата потребовалось бы четыре выражения и переменная. Следующие четыре выражения, разделенные точками, эквивалентны приведенному выше каскадному выражению.

```
temp ← OrderedCollection new.
temp add: 1.
temp add: 2.
temp add: 3
```

Блоки

Блоки — объекты, используемые во многих управляющих структурах системы Smalltalk-80. Блок представляет собой отложенную последовательность действий. Выражение-блок состоит из заключенной в квадратные скобки последовательности выражений, разделяемых точками. Например,

```
[index ← index + 1]
```

или

```
[index ← index + 1.
 array at: index put: 0]
```

Если точка стоит после последнего выражения блока, то она игнорируется, как, например, в блоке

```
[expenditures at: reason.]
```

Когда блок встречается в выражении, заключенные в квадратные скобки выражения блока сразу не выполняются. Значение выражения-блока — объект, который может выполнить эти вложенные выражения позже, когда ему будет дан запрос сделать это. Например, выражение

```
action at: 'monthly payments'
        put: [HouseholdFinances spend: 650 for: 'rent'.
```

```
HouseholdFinances spend: 7.25 for: 'newspaper'.
HouseholdFinances spend: 225.50 for: 'car payment']
```

в действительности не посылает ни одного из трех сообщений spend:for: к HouseholdFinances. Оно, просто устанавливает связь блока со строкой 'monthly payments'.

Последовательность действий, которую описывает блок, будет выполнена только тогда, когда блок получит унарное сообщение value (значение). Например, следующие два выражения дадут один и тот же результат:

```
index ← index + 1
```

и

```
[index ← index + 1] value
```

Объект, описываемый выражением

```
action at: 'monthly payments'
```

(см. выше) — это блок, состоящий их трех сообщений spend:for:. Выполнение выражения

```
(action at: 'monthly payments') value
```

приведет к посылке трех сообщений spend:for: объекту HouseholdFinances.

Блок также может быть присвоен переменной. Если выполнить выражение

```
incrementBlock ← [index ← index + 1]
```

то выражение

```
incrementBlock value
```

увеличит index на 1.

Объект, возвращаемый после выполнения посланного блоку сообщения value, представляет значение последнего выполненного выражения блока. Так, если выполнено выражение

```
addBlock ← [index + 1]
```

то еще один способ увеличить index на 1, это выполнить

```
index ← addBlock value
```

Блок, который не содержит никаких выражений, возвращает nil, когда ему посылается сообщение value. То есть, выражение

```
[] value
```

имеет значение nil.

Управляющие структуры

Управляющие структуры определяют порядок некоторых действий. Фундаментальная управляющая структура в языке Smalltalk-80 обеспечивает последовательное выполнение следующих друг за другом выражений языка. Многие управляющие структуры, нарушающие последовательное выполнение выражений, могут быть реализованы с помощью блоков. Эти управляющие структуры вызываются либо посылкой сообщения блоку, либо посылкой сообщения с одним или более аргументами-блоками. Ответ на одно из этих сообщений и определяет порядок дальнейших действий посредством посылки блоку (или блокам) сообщения value.

Рассмотрение хода выполнения следующей последовательности выражений дает пример работы блоков.

```
incrementBlock ← [index ← index + 1].
sumBlock ← [sum + (index * index)].
sum ← 0.
index ← 1.
sum ← sumBlock value.
```

```
incrementBlock value.
sum ← sumBlock value
```

В результате выполнения этой последовательности выражений будут выполнены следующие 15 действий:

1. *Присвоить* блок переменной incrementBlock.
2. *Присвоить* блок переменной sumBlock.
3. *Присвоить* число 0 переменной sum.
4. *Присвоить* число 1 переменной index.
5. *Послать* сообщение value блоку sumBlock.
6. *Послать* сообщение * 1 числу 1.
7. *Послать* сообщение + 1 числу 0.
8. *Присвоить* число 1 переменной sum.
9. *Послать* сообщение value блоку incrementBlock.
10. *Послать* сообщение + 1 числу 1.
11. *Присвоить* число 2 переменной index.
12. *Послать* сообщение value блоку sumBlock.
13. *Послать* сообщение * 2 числу 2.
14. *Послать* сообщение + 4 числу 1.
15. *Присвоить* число 5 переменной sum.

Пример управляющей структуры, реализованной с помощью блоков, — простое повторение действий, представленное посылкой целому числу сообщения timesRepeat: с блоком в качестве аргумента. В ответ целое число посылает блоку-аргументу сообщение value столько раз, сколько единиц содержит значение числа. Следующее выражение четыре раза удвоит значение переменной с именем amount:

```
4 timesRepeat: [amount ← amount + amount]
```

Условные выражения

Две общие управляющие структуры, реализованные с помощью блоков, — *условный выбор* и *условное повторение*. Условный выбор похож на оператор if-then-else в алголоподобных языках, а условное повторение похоже на операторы while и until в таких языках. Эти условные управляющие структуры используют два логических объекта true и false, описанных в разделе “Псевдопеременные”. Логические объекты возвращаются после посылок сообщений, которые требуют простого ответа “да-нет” (например, сообщений для сравнения величин: <, <=, >, >=, ~=).

☐ *Условный выбор* Условный выбор действий обеспечивается сообщением к логическим объектам с именем сообщения ifTrue:ifFalse: (еслиИстина:еслиЛожь:) и двумя аргументами-блоками. Единственные объекты, которые понимают сообщение ifTrue:ifFalse:, — объекты true и false. Они выполняют противоположные действия: true посылает сообщение value первому аргументу-блоку и игнорирует второй аргумент, false посылает сообщение value второму аргументу-блоку и игнорирует первый аргумент. Например, следующее выражение присваивает 0 или 1 переменной parity, в зависимости от того, делится на 2 значение переменной number или нет. Бинарное сообщение \\ вычисляет остаток от деления целых чисел.

```
(number \\ 2) = 0
    ifTrue: [parity ← 0]
    ifFalse: [parity ← 1]
```

Значение, возвращенное из ifTrue:ifFalse:, это значение того блока, который был выполнен. Предыдущий пример может быть записан и по другому:

```
parity ← (number \\ 2) = 0 ifTrue: [0] ifFalse: [1]
```

В дополнение к сообщению ifTrue:ifFalse: есть два однословных ключевых сообщения, определяющих только одно условное следствие. Имена этих сообщений ifTrue: и ifFalse:. Они дают тот же результат, что и сообщение ifTrue:ifFalse:, когда один из его аргументов — пустой блок. Например, следующие два выражения дают одинаковый результат:

```
index <= limit
    ifTrue: [total ← total + (list at: index)]
```

и

```
index <= limit
    ifTrue: [total ← total + (list at: index)]
    ifFalse: []
```

Так как значение пустого блока равно nil, следующее выражение устанавливает lastElement в nil, если index больше чем limit.

```
lastElement ← index > limit ifFalse: [list at: index]
```

□ *Условное повторение* Условное повторение некоторой последовательности действий обеспечивается посылкой блоку сообщения с именем whileTrue: (покаИстина:) и другим блоком в качестве аргумента. Блок-получатель посылает себе сообщение value и, если ответ равен true, посылает аргументу-блоку сообщение value, после чего начинает все сначала, посылая себе сообщение value и т. д. Когда ответ блока-получателя на сообщение value становится false, он останавливает повторение и выходит из сообщения whileTrue:. Например, условное повторение может быть использовано для инициализации всех элементов массива с именем list.

```
index ← 1.
[index <= list size]
    whileTrue: [list at: index put: 0.
            index ← index + 1]
```

Блоки также понимают сообщение с именем whileFalse: (покаЛожь:), которое повторяет выполнение блока-аргумента, пока значение блока-получателя остается false. Поэтому следующие выражения эквивалентны приведенным выше выражениям:

```
index ← 1.
[index > list size]
    whileFalse: [list at: index put: 0.
            index ← index + 1]
```

Программист волен выбирать, какое сообщение использовать. Значение, возвращаемое сообщениями whileTrue: и whileFalse:, всегда nil. Два дополнительных сообщения, whileTrue и whileFalse, эквивалентны, соответственно, whileTrue: [] и whileFalse: [].

Аргументы блока

Для того, чтобы некоторые непоследовательные управляющие структуры, было легче реализовать, блоки могут иметь один или более аргументов. Аргументы определяются включением в начало блока идентификаторов, каждому из которых прошествует двоеточие. Аргументы блока отделяются от выражений, составляющих блок, вертикальной чертой. Следующие два примера описывают блоки с одним аргументом:

```
[:array | total ← total + array size]
```

и

```
[:newElement |
 index ← index + 1.
 list at: index put: newElement]
```

Типичное использование блоков с аргументами — реализация функций, применяемых ко всем элементам некоторой структуры данных. Например, многие объекты, представляющие собой различные виды структур данных отвечают на сообщение do: (выполнить:), которое

имеет аргументом блок с одной переменной. Объект-получатель, получив сообщение do:, выполняет блок-аргумент один раз для каждого своего элемента. Каждый элемент из получателя становится значением аргумента блока в течении одного выполнения блока. Следующий пример вычисляет сумму квадратов первых пяти простых чисел. Результат будет значением переменной sum.

```
sum ← 0.
#(2 3 5 7 11) do: [:prime | sum ← sum + (prime * prime)]
```

Сообщение collect: (собрать:) создает набор значений, возвращаемых блоком, которому поставляются элементы получателя. Значение следующего выражения — массив квадратов первых пяти простых чисел.

```
#(2 3 5 7 11) collect: [:prime | prime * prime]
```

Объекты, которые реализуют эти управляющие структуры, поставляют значения аргументам блока, посылая этому блоку сообщения value:. Блок с одним аргументом, получив сообщение value:, присваивает аргументу блока значение аргумента сообщения value: и последовательно выполняет все выражения блока. Например, выполнение следующих выражений приведет к тому, что переменная total будет иметь значение 7:

```
sizeAdder ← [:array | total ← total + array size].
total ← 0.
sizeAdder value: #(a b c).
sizeAdder value: #(1 2).
sizeAdder value: #(e f)
```

Блоки могут иметь и более одного аргумента. Все аргументы блока локальны по отношению к блоку и его выполнению. Например,

```
[:x :y | (x * x) + (y * y)]
```

или

```
[:frame :clippingBox | frame intersect: clippingBox]
```

Блок должен иметь столько аргументов, каково число ключевых слов value: в посылаемом ему сообщении. Два приведенные выше блока могут быть выполнены посредством посылки им сообщения с именем value:value:. Два аргумента этого сообщения по порядку задают значения двум аргументам блока-получателя. Если блок получает сообщение с количеством ключевых слов value: отличным от числами аргументов блока, то будет выдано сообщение об ошибке.

Словарь терминов

Синтаксис выражений кратко изложен на последней странице обложки этой книги.

выражение — последовательность символов, описывающая объект.

литерал — выражение, описывающее константу, такую как число или строка.

имя — строка, чья последовательность символов гарантирует отличие от любого другого имени.

массив — структура данных, элементы которой индексированы целыми числами.

имя переменной — выражение, описывающее текущее значение переменной.

присваивание — выражение, описывающее изменение значения переменной.

псевдопеременная	выражение похожее на имя переменной. Однако, в отличие от имени переменной, значение псевдопеременной не может быть изменено присваиванием.
получатель	объект, которому посылается сообщение.
имя сообщения	имя типа операции, которую сообщение запрашивает у получателя.
аргумент сообщения	объект, который определяет дополнительную информацию для выполнения операции.
унарное сообщение	сообщение без аргументов.
ключевое слово	идентификатор с последующим двоеточием.
ключевое сообщение	сообщение с одним или более аргументами, имя которого состоит из одного или нескольких ключевых слов.
бинарное сообщение	сообщение с одним аргументом, имя которого состоит из одного или двух специальных символов.
каскадирование	описание в одном выражении нескольких сообщений одному объекту.
блок	описание отложенной последовательности действий.
аргумент блока	параметр, который должен быть предоставлен блоку при его выполнении.
value	сообщение блоку, запрашивающее выполнение его выражений.
value:	ключевое слово в сообщении блоку с одним аргументом, запрашивающее выполнение его выражений.
ifTrue:ifFalse:	сообщение логическому объекту, запрашивающее условный выбор.
ifFalse:ifTrue:	сообщение логическому объекту, запрашивающее условный выбор.
ifTrue:	сообщение логическому объекту, запрашивающее условный выбор.
ifFalse:	сообщение логическому объекту, запрашивающее условный выбор.
whileTrue:	сообщение блоку, запрашивающее условное повторение.
whileFalse:	сообщение блоку, запрашивающее условное повторение.
whileTrue	сообщение блоку, эквивалентное whileTrue: [].
whileFalse	сообщение блоку, эквивалентное whileFalse: [].
do:	сообщение, посылаемое набору для перечисления его элементов.
collect:	сообщение, посылаемое набору для преобразования его элементов.