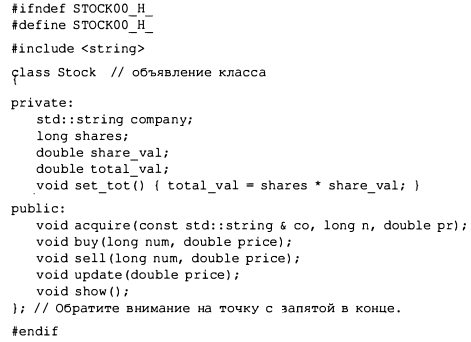
## Конструкторы и деструкторы классов

Конструктором называется метод, имя которого совпадает с именем класса. Конструктор предназначен для инициализации объекта.

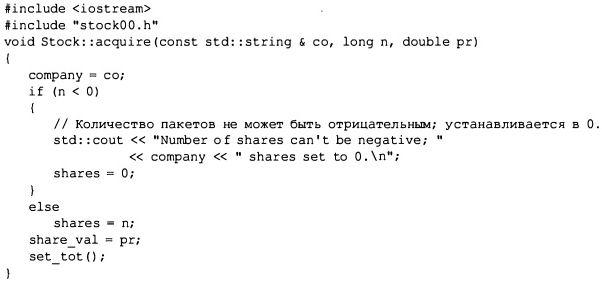
Деструктор — это особый вид метода, применяющийся для освобождения памяти, занимаемой объектом. Деструктор вызывается автоматически, когда объект выходит из области видимости

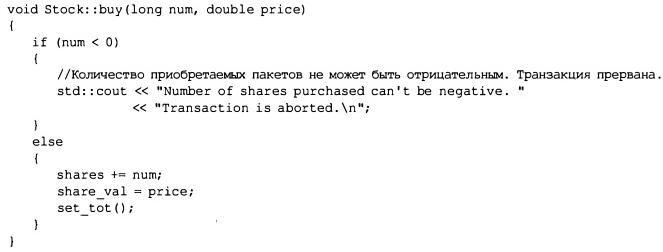
В предыдущем параграфе был рассмотрен пример, «моделирующий» покупку, продажу и т.д. пакетов акций компаний. В нем не были явно описаны конструкторы и деструкторы. Приведем еще раз примеры 1.1 – 1.3.

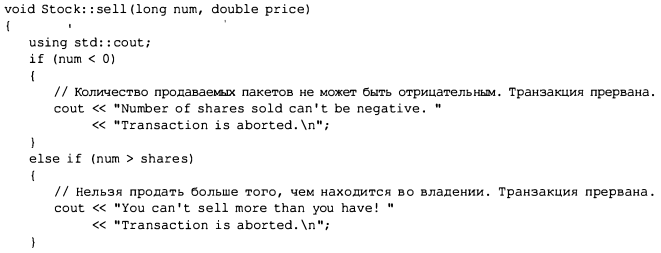
Пример 1.1 (stock00.h - файл интерфейса).

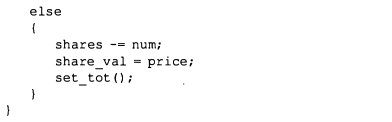


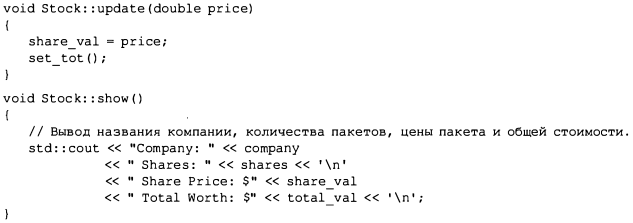
Пример 1.2 (stocks00.cpp - файл реализации).



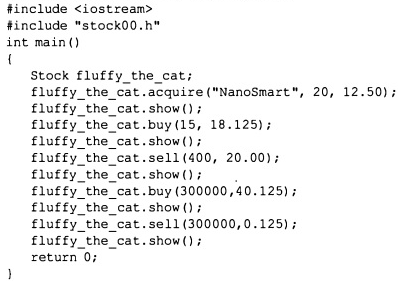




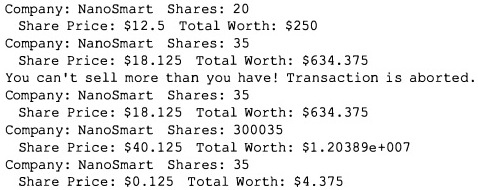




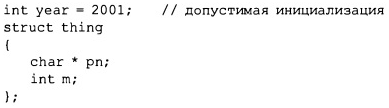
Пример 1.3 (usestok0.cpp).



Результат работы программы:



В классе должны присутствовать конструкторы и деструкторы. Одна из целей C++ состоит в том, чтобы сделать использование объектов классов подобным применению стандартных типов. Однако код, который был приведен до сих пор, не позволяет инициализировать объект Stock таким же способом, как это можно сделать с int или struct. T. е. обычный синтаксис инициализации не применим к типу Stock. Например:





Причина, по которой нельзя таким способом инициализировать объект Stock, связана с тем, что к данным класса разрешен только закрытый доступ, а это означает, что единственный способ, с помощью которого программа может получить доступ к ним — через функции-члены. Следовательно, для успешной инициализации объекта понадобится придумать соответствующую функцию-член. Чтобы инициализировать объект класса так, как показано выше, нужно объявить данные-члены как public, a не private, но в этом случае нарушается один из базовых принципов использования классов — сокрытие данных.

В общем случае лучше, чтобы все объекты инициализировались при их создании. Например, рассмотрим следующий код:



При существующей реализации класса Stock объект gift не имеет установленного значения для члена company. В проектном решении класса предполагается, что пользователь вызовет метод acquire () раньше любых других функций-членов, однако нет какого-либо средства, чтобы гарантировать это. Единственным способом обойти эту трудность является автоматическая инициализация объектов при их создании. Для этого в C++ предлагаются специальные функции-члены, называемые конструкторами класса, которые предназначены для создания новых объектов и присваивания значений их членам-данным. Если говорить точнее, то C++ регламентирует имя для таких функций-членов, а также синтаксис их вызова, тогда как ваша задача — написать определение этого метода. Имя метода конструктора совпадает с именем класса. Например, возможный конструктор для класса Stock — это функция-член Stock(). Прототип и заголовок конструктора обладают интересным свойством: несмотря на то, что конструкторы не имеют возвращаемого значения, они не объявляются с типом void. Фактически конструкторы не имеют объявленного типа.

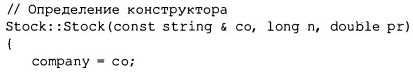
## Объявление и определение конструкторов

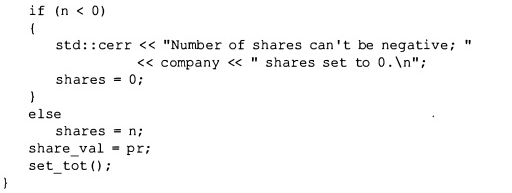
Поскольку объект Stock имеет три значения, которые ему нужно получить из «внешнего мира», то необходимо передать конструктору три аргумента. Четвертое значение — это член total\_val; он вычисляется на основе shares и share\_val, поэтому передавать его конструктору не понадобится. Возможно, вы решите передать только значение члена company и установить остальные члены в нули; это можно сделать с использованием аргументов по умолчанию. Таким образом, прототип конструктора будет выглядеть следующим образом:



Первый аргумент представляет собой указатель на строку, используемую для инициализации члена класса company типа string. Аргументы n и pr предоставляют значения для членов shares и share\_val. Обратите внимание, что тип возвращаемого значения конструктора не указан. Прототип размещен в открытом разделе объявления класса.

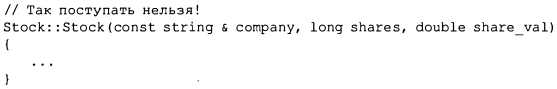
Ниже приведен один из вариантов определения конструктора:





Это тот же код, который использовался ранее для функции acquire(). Разница в том, что в данном случае программа автоматически вызовет конструктор при объявлении объекта.

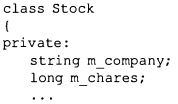
**Замечание** (имена параметров в конструкторах). Часто присутствуют попытки использовать имена переменных-членов класса в качестве имен аргументов в конструкторе, как показано в следующем примере:



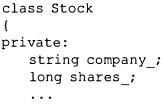
Это неверно. Аргументы конструктора не являются переменными-членами; они представляют значения, которые присваиваются членам класса. Таким образом, они должны иметь отличающиеся имена, иначе вы столкнетесь с непонятным кодом вроде такого:



Одним из часто используемых способов, призванных помочь избежать этого, является использование префикса m\_ для идентификации данных-членов:



Другой также часто применяемый способ заключается в применении суффикса в виде подчеркивания для имен членов:



## Использование конструкторов

Язык C++ предлагает два способа инициализации объектов с помощью конструктора. Первый — вызвать конструктор явно:



Здесь устанавливается значение членов объекта food: значение company = "World Cabbage", значение shares = 250, share\_val = 1.25.

Второй способ — вызвать конструктор неявно:



Эта более компактная форма эквивалентна следующему явному вызову:



C++ использует конструктор класса всякий раз, когда вы создаете объект класса, даже если применяется операция new для динамического выделения памяти. Ниже показано, как использовать конструктор вместе с new:



Оператор, создающий объект Stock, инициализирует его значениями, переданными в аргументах, и присваивает адрес нового объекта указателю pstock. В этом случае объект не имеет имени, но для управления объектом можно применять указатель. Указатели на объекты будут обсуждаться позже.

Конструктор с объектом взаимодействует иначе, чем объект с остальными методами класса. Обычно объект применяется для вызова метода:



Однако нельзя использовать объект для вызова конструктора, поскольку до тех пор, пока конструктор не завершит создание объекта, его не существует. Вместо того чтобы вызываться объектом, конструктор служит для создания объекта.

## Конструкторы по умолчанию

Конструктор по умолчанию — это конструктор, использующийся для создания объекта, но в котором не предоставлены явные инициализирующие значения. То есть это конструктор, который применяется для объявлений, подобных показанному ниже:



Но ведь в примере 1.3 уже делалось это! Причина, по которой этот оператор работает, состоит в том, что если вы забудете о написании конструкторов, то C++ автоматически создаст конструктор по умолчанию. Это — неявная версия конструктора по умолчанию, который ничего не делает. Для класса Stock конструктор по умолчанию будет таким:



В результате создается объект fluffy\_the\_cat с инициализированными членами, как в следующем операторе создается х без указания его значения:



То, что конструктор по умолчанию не имеет аргументов, отражает факт отсутствия значений в объявлении. Любопытным моментом, имеющим отношение к конструктору, но умолчанию, является то, что компилятор создает его, только если вы не определите ни одного собственного конструктора. После того, как вы определите хотя бы один конструктор класса, компилятор перестанет создавать конструктор по умолчанию. Если вы предоставите конструктор не по умолчанию, например, Stock (const string & со, long n, double pr), но не предложите собственную версию конструктора по умолчанию, то следующее объявление вызовет ошибку:



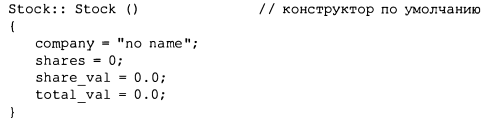
Причина такого поведения в том, что может понадобиться запрет создания неинициализированных объектов. Если же, однако, вы предпочитаете создавать объекты без явной инициализации, то должны будете определить собственный конструктор. Этот конструктор не имеет аргументов. Конструктор по умолчанию можно создать двумя способами. Один из них предусматривает указание значений по умолчанию для всех аргументов в существующем конструкторе:



Второй способ — использование возможности перегрузки функций для определения второго конструктора без аргументов:



Допускается наличие только одного конструктора по умолчанию, поэтому удостоверьтесь, что не создали их два. На самом деле вы должны инициализировать объекты для гарантии того, что при создании объекта все члены получают известные и подходящие значения. Таким образом, пользовательский конструктор по умолчанию, как правило, обеспечивает явную инициализацию всех переменных-членов. Например, ниже показано, как можно определить конструктор по умолчанию для класса Stock:



После создания конструктора по умолчанию любым из двух способов (без аргументов или с предоставлением значений по умолчанию для всех аргументов) можно объявлять объектные переменные без явной инициализации:



Однако вас не должна сбивать с толку неявная форма конструктора не по умолчанию:



Первое объявление из приведенных выше вызывает конструктор не по умолчанию — т.е. такой, который принимает аргументы. Второе объявление устанавливает, что second() — это функция, возвращающая объект Stock. При неявном вызове конструктора по умолчанию круглые скобки указываться не должны.

## Деструкторы

В случае использования конструктора для создания объекта программа отслеживает этот объект до момента его исчезновения. В этот момент программа автоматически вызывает специальную функцию-член под названием деструктор. Деструктор призван очищать всякий информационный "мусор". Например, если в конструкторе используется операция new для выделения памяти, то деструктор должен обратиться к delete для ее освобождения. Конструктор нашего класса Stock не делает вызова операции new, поэтому деструктор для класса Stock явно использовать не нужно. В таком случае вы можете просто позволить компилятору сгенерировать неявный деструктор, который ничего не делает, что и имеет место в первой версии класса Stock. С другой стороны, необходимо знать, как объявляются и определяются деструктор, поэтому давайте предусмотрим его в классе Stock.

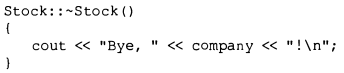
Как и конструктор, деструктор имеет специальное имя. Оно формируется из имени класса и предваряющего его символа тильды (~). То есть деструктором класса Stock называется ~Stock(). Подобно конструктору, деструктор не имеет ни возвращаемого значения, ни объявляемого типа. Однако в отличие от конструктора, деструктор не должен иметь аргументы. Таким образом, прототип деструктора класса Stock выглядит следующим образом:



Поскольку деструктор Stock не имеет никаких обязанностей, его можно кодировать как функцию, которая ничего не делает:



Но просто для того, чтобы увидеть, когда вызывается деструктор, определим его следующим образом:



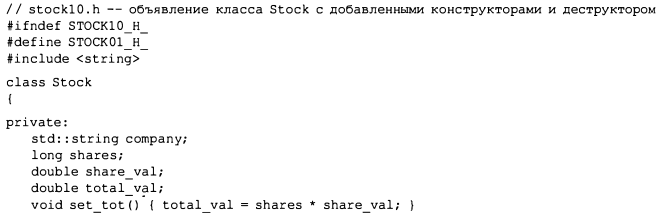
Когда должен вызываться деструктор? Этим управляет компилятор. Обычно деструктор не должен явно вызываться в коде.

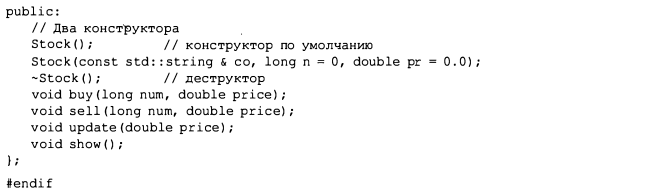
## Усовершенствование класса Stock

Теперь необходимо включить конструкторы и деструктор в определение класса и методов. Учитывая важность добавления конструкторов, изменим имя stock00.h на stock10.h. Методы класса будут находиться в файле по имени stock10.cpp. И, наконец, программа, использующая эти ресурсы, будет помещена в третий файл — usestok2.срр.

В примере 2.1 показан заголовочный файл программы Stock. К исходному объявлению класса здесь добавлены прототипы функций конструктора и деструктора. Также он отличается отсутствием функции acquire(), которая более не нужна, поскольку класс имеет конструкторы. В файле используется также прием #ifndef, для защиты от многократного включения заголовочного файла.

Пример 2.1 (stock10.h).



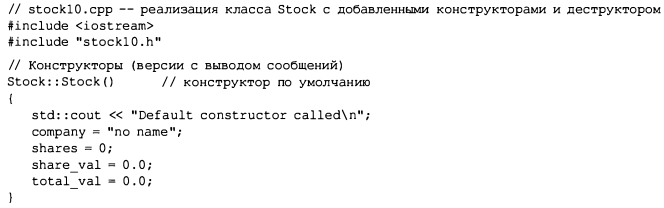


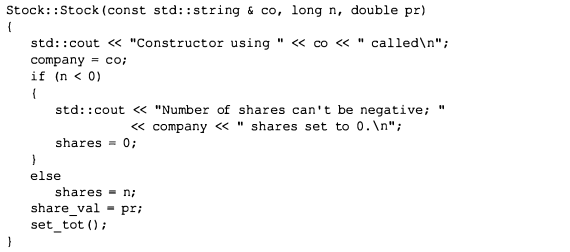
## Файл реализации

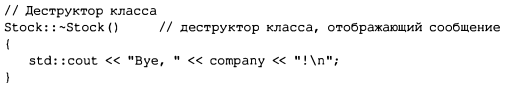
В примере 2.2 приведены определения методов для разрабатываемой программы. Для предоставления программе необходимого объявления класса в нем включается файл stock10.h. (Вспомните, что помещение имени файла в двойные кавычки вместо угловых скобок заставляет компилятор искать его там же, где расположены файлы исходного кода.)

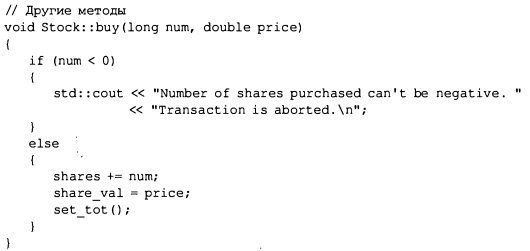
Напомним, что включение заголовочного файла iostream необходимо для обеспечения поддержки ввода-вывода. В коде также демонстрируется использование объявлений using и уточненных имен (наподобие std::string) для обеспечения доступа к различным определениям из заголовочных файлов. В этом файле к предшествующим методам добавлены определения методов конструктора и деструктора. Чтобы помочь увидеть момент вызова метода, каждый из них отображает сообщение. Это не является обычным свойством конструкторов и деструкторов, однако позволяет сделать наглядным их использование классом.

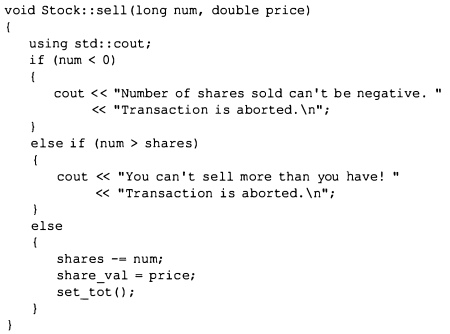
Пример 2.2 (stock10.cpp).

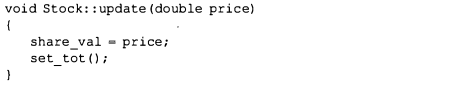


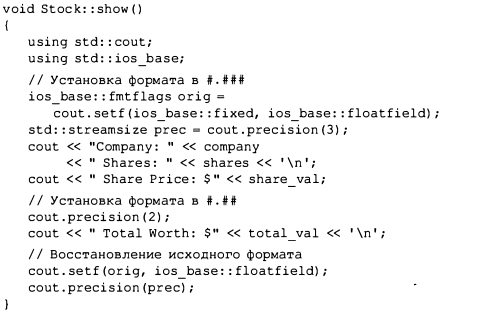








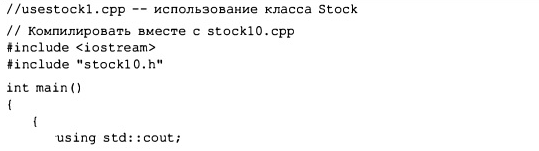


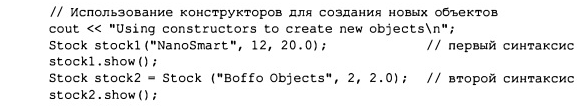


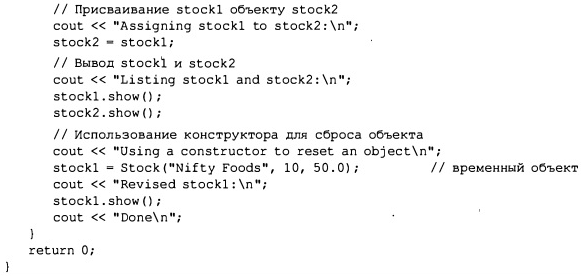
## Файл клиентской программы

В примере 2.3 представлена программа для тестирования новых методов. Из-за того, что здесь просто используется класс Stock, код, приведенный в листинге, является клиентом класса Stock. Как и stock10.cpp, он включает файл stock10.h для доступа к объявлению класса. Эта программа демонстрирует работу конструкторов и деструктора. В ней также используются команды форматирования.

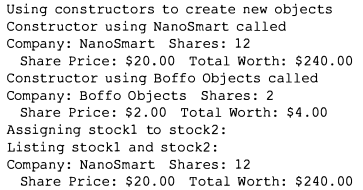
Пример 2.3 (usestock1.срр). Клиентская программа.

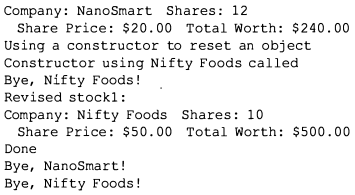






Вывод программы:





**Замечание.** Вы наверняка заметили, что в примере 2.3 присутствуют дополнительные фигурные скобки в начале и почти в конце функции main (). Срок существования автоматических переменных, таких как stock1 и stock2, истекает, когда выполнение программы покидает блок, содержащий их определения. Без этих дополнительных фигурных скобок таким блоком являлось бы тело функции main (), поэтому деструкторы не были бы вызваны вплоть до полного завершения main (). В оконной среде это означает, что окно программы закроется перед вызовом двух деструкторов, и увидеть два последних сообщения не удастся. Но благодаря скобкам, два последних вызова деструкторов произойдут перед достижением оператора return, так что сообщения смогут отобразиться.

В примере 2.3 оператор



создает объект Stock по имени stock1 и инициализирует его данные-члены указанными значениями:



Следующий оператор использует другой синтаксис для создания и инициализации объекта stock2:



Стандарт C++ позволяет компилятору выполнять второй синтаксис двумя способами. Один из них характеризуется тем же поведением, что и в случае первого синтаксиса:



Второй способ реализации заключается в вызове конструктора для создания временного объекта, который затем копируется в stock2. После этого временный объект уничтожается. Если компилятор использует этот вариант, то для временного объекта вызывается деструктор, что приводит к следующему выводу:



Компилятор, который обеспечивает генерацию такого вывода, обычно освобождает временный объект немедленно, но может быть и так, что он ожидает некоторое время — в этом случае сообщение из деструктора появится позже.

Приведенный ниже оператор иллюстрирует возможность присваивания одного объекта другому объекту того же типа:

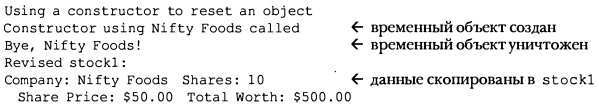


Как и в случае присваивания структур, присваивание объектов класса по умолчанию копирует члены одного объекта в другой. В этом случае исходное содержимое stock2 перезаписывается.

Конструктор можно использовать не только для инициализации нового объекта. Например, в функции main () присутствует такой оператор:



Объект stock1 уже существует. Следовательно, вместо инициализации объекта stock1 показанный оператор присваивает ему новые значения. Это делается за счет создания конструктором нового временного объекта и последующего копирования его содержимого в stock1. Затем программа уничтожает временный объект, вызывая его деструктор, что и иллюстрирует следующий вывод:



Некоторые компиляторы могут освобождать временный объект позже, откладывая вызов деструктора.

В конце работы программа отображает следующие сообщения:



Когда функция main() завершает работу, ее локальные переменные (stock1 и stock2) перестают существовать. Поскольку такие автоматические переменные размещаются в стеке, последний созданный объект удаляется первым, а первый созданный — последним. Вспомним, что строка "NanoSmart" находилась изначально в stock1, но позже была перенесена в stock2, а объект stock1 был сброшен в "Nifty Foods".

Вывод программы демонстрирует фундаментальную разницу между следующими двумя операторами:



Первый из этих операторов вызывает инициализацию; он создает объект с указанным значением, и может создавать либо не создавать временный объект. Второй оператор вызывает присваивание. Использование конструктора в операции присваивания в таком виде всегда служит причиной создания временного объекта перед выполнением собственно присваивания.

## Списковая инициализация С++11

В стандарте C++11 можно использовать синтаксис списковой инициализации для классов. Для этого потребуется предоставить в фигурных скобках содержимое, соответствующее списку аргументов конструктора:



Списки в фигурных скобках в первых двух объявлениях соответствуют следующему конструктору:



Таким образом, этот конструктор будет использоваться для создания двух объектов. В случае объекта jock для второго и третьего аргументов будут применяться значения по умолчанию — 0 и 0.0. Третье объявление соответствует конструктору по умолчанию, поэтому объект temp будет создан с его помощью.

Отметим,что С++11 предлагает класс по имени std::initializer\_list, который может использоваться в качестве типа для параметра функции или метода. Этот класс представляет список произвольной длины, все элементы которого имеют один и тот же тип или могут быть преобразованы к одному и тому же типу.