Современное проектирование на C++

Обобщенное программирование и прикладные шаблоны проектирования

Андрей Александреску

Издательский дом “Вильямс”
Москва • Санкт-Петербург • Киев
2008

Modern C++ Design

Generic Programming
and Design Pattern Applied

Andrei Alexandrescu

ADDISON–WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico • City

Современное проектирование на C++

ББК 32.973.26-018.2.75
А46
УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией *А.В. Слепцов*

Перевод с английского и редакция канд.физ.-мат.наук *Д.А. Клюшина*

По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Александреску, Андрей.

А46 Современное проектирование на C++. Серия *C++ In-Depth*. : Пер. с англ. — М. : Издательский дом "Вильямс", 2008. — 336 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-0351-8 (рус.)

В книге изложена новая технология программирования, представляющая собой сплав обобщенного программирования, шаблонного метапрограммирования и объектно-ориентированного программирования на C++. Обобщенные компоненты, созданные автором, высоко подняли уровень абстракции, наделив язык C++ чертами языка спецификации проектирования, сохранив всю его мощь и выразительность.

В книге изложены способы реализации основных шаблонов проектирования. Разработанные компоненты воплощены в библиотеке Loki, которую можно загрузить с Web-страницы автора. Книга предназначена для опытных программистов на C++.

ББК 32.973.26-018.2.75

ISBN 978-5-8459-0351-8 (рус.)
ISBN 0-201-77581-6 (англ.)

Оглавление

Содержание

Предисловие Скотта Мейерса

В 1991 году вышло первое издание книги "Эффективное использование C++" ("Effective C++"). В нем почти не рассматривались шаблоны, поскольку в то время они были новшеством, и я о них практически ничего не знал. Включенные в книгу немногочисленные фрагменты программ, содержащих шаблоны, я был вынужден посылать по электронной почте другим людям, поскольку все доступные мне компиляторы их не поддерживали.

В 1995 году я написал книгу "Наиболее эффективное использование C++" ("More effective C++"). И снова почти не упомянул о шаблонах. На этот раз меня остановило не отсутствие знаний (в первоначальном варианте книга содержала целую главу, посвященную этой теме) и не ограниченность моих компиляторов. Просто возникло подозрение, что понимание роли шаблонов в среде программистов на языке C++ претерпевает настолько значительные изменения, что все мои мысли по этому поводу могут быстро стать банальными, поверхностными и даже ошибочными.

Эти подозрения возникли по двум причинам. Первой из них была статья, опубликованная в январском номере журнала "C++ Report" за 1995 год Джоном Бартоном (John Barton) и Ли Нэкманом (Lee Nackman). В ней описывалось применение шаблонов для безопасного анализа размерностей (typesafe dimension analysis) без дополнительных затрат машинного времени. Я сам довольно долго пытался решить эту задачу и знал, что другие программисты также безуспешно ломают над ней голову. Революционный подход, предложенный Бартоном и Нэкменом, помог мне понять, что с помощью шаблонов можно не просто создавать контейнеры, содержащие объекты класса `T`, но и достичь намного более значительных результатов.

В качестве иллюстрации этого подхода рассмотрим код, предназначенный для умножения двух физических величин произвольной размерности.

```
template<int m1, int l1, int t1, int m2, int l2, int t2>
Physical<m1+m2, l1+l2, t1+t2> operator*(Physical<m1, l1, t1> lhs,
                                        Physical<m2, l2, t2> rhs)
{
   return  Physical<m1+m2,l1+l2,t1+t2>::
           unit*lhs.value()*rhs.value();
}
```

Даже без объяснений, приведенных в статье, совершенно очевидно, что эта шаблонная функция (function template) получает шесть параметров, ни один из которых не представляет собой какой-либо тип! Это явилось для меня приятным открытием.

Вскоре я стал изучать стандартную библиотеку шаблонов STL (Standard Templates Library). Эта разработка Александра Степанова (Alexander Stepanov) весьма элегантна. В ней контейнеры ничего не знают об алгоритмах, алгоритмы ничего не знают о контейнерах, итераторы функционируют как указатели (но могут быть объектами), контейнеры и алгоритмы одинаково успешно могут получать указатели на функции и сами функции в виде объектов, а пользователи библиотеки могут расширять ее, не прибегая к наследованию от какого-либо базового класса или переопределению виртуальных функций. И тут я почувствовал (как и при чтении статьи Бартона и Нэкмена), что практически *ничего* не знаю о шаблонах.

По этой причине я не стал почти ничего писать о шаблонах в книге "Наиболее эффективное использование C++". Как я мог касаться этой темы, если мое понима-

ние шаблонов оставалось на уровне контейнеров, содержащих объекты класса `T`, в то время как Бартон, Нэкман, Степанов и другие продемонстрировали, что такое применение шаблонов является лишь вершиной айсберга?

В 1998 году Андрей Александреску (Andrei Alexandrescu) и я стали обмениваться сообщениями по электронной почте, и вскоре я понял, что мне придется снова изменить свое мнение о шаблонах. В то время как Бартон, Нэкман и Степанов ошеломили меня тем, *что* можно сделать с помощью шаблонов, Андрей поразил меня, объяснив, *как* это можно сделать.

Одна из простейших вещей, которые он помог мне изложить в общедоступной форме, до сих пор остается примером, который я первым привожу людям, начинающим работать в этой области. Это шаблон `CTAssert`, представляющий собой аналог макроса `assert`, но позволяющий проверять условия во время компиляции, а не во время выполнения программы.

```
template<bool> struct CTAssert;
template<> struct CTAssert<true> {};
```

Вот и все! Обратите внимание на то, что обычный шаблон `CTAssert` нигде не определяется. Более того, он конкретизирован только для значения `true`, но не для `false`. То, что есть в этом шаблоне, не менее важно, чем то, *чего в нем нет*. Это заставляет посмотреть на код этого шаблона под другим углом, поскольку оказывается, что бо́льшая часть его "исходного кода" осознанно проигнорирована. Этот образ мышления совершенно отличается от общепринятого. (В этой книге Андрей обсуждает более сложный шаблон `CompileTimeCheker`.)

В итоге Андрей приступил к разработке шаблонно-ориентированной реализации распространенных языковых идиом (language idioms) и шаблонов проектирования, особенно шаблонов GoF[1]. Это вызвало перепалку в среде разработчиков шаблонов, поскольку они были абсолютно убеждены, что эти шаблоны невозможно запрограммировать. Когда стало ясно, что Андрей создал средства для автоматического генерирования *реализаций* шаблонов, а не для программирования собственно шаблонов, возражения были сняты. Мне было приятно узнать, что Андрей и один из разработчиков шаблонов GoF (Джон Влиссидес) вместе написали две статьи в журнале "C++ Report", посвященные этой теме.

Следуя выбранному направлению, связанному с шаблонами для генерации идиом и реализациями шаблонов проектирования, Андрей столкнулся с проблемами, стоящими и перед другими программистами, работающими в этой области. Должна ли программа быть безопасной в многопоточной среде (thread safe)? Откуда брать дополнительную память: из кучи, стека или пула статической памяти? Нужно ли перед разыменованием интеллектуальных указателей (smart pointers) проверять, равны ли они нулю? Что случится при завершении программы, если один деструктор синглтона (singlton's destructor) попытается использовать уже уничтоженный синглтон? Андрей стремился предложить пользователям максимально широкий выбор возможностей, не навязывая своего мнения.

Он решил инкапсулировать эти решения в виде *классов стратегий* (policy classes), что позволило пользователям передавать их как шаблонные параметры. Кроме того, Андрей предложил для этих классов разумные значения по умолчанию, так что боль-

---

[1] Название этих шаблонов происходит от словосочетания "Gang of Four" — "Банда четырех". В состав этой "банды" входили Erich Gamma (Эрих Гамма), Richard Helm (Ричард Хелм), Ralph Johnson (Ральф Джонсон) и John Vlissides (Джон Влиссидес), написавшие основополагающую книгу *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1995).

шинство клиентов может их просто игнорировать. Результат оказался потрясающим! Например, шаблон для интеллектуального указателя, описанный в книге, получает в виде параметров только 4 класса стратегий, а генерирует более 300 разных типов интеллектуальных указателей, каждый из которых обладает уникальными особенностями поведения. Программисты, осведомленные о поведении интеллектуального указателя, заданном по умолчанию, могут вообще проигнорировать параметры, представляющие собой классы стратегий, указав лишь тип объекта, на который он должен ссылаться. Это позволяет им извлекать выгоду из прекрасно сделанного класса для интеллектуального указателя, не прилагая никаких усилий.

В конце книги рассмотрены три разные темы, причем для каждой из них выбран свой способ изложения. Во-первых, представлен новый взгляд на мощь и гибкость шаблонов в языке C++. (Если, прочитав материал, посвященный спискам типов (typelists), вы не свалились со стула, значит, вы сидели на полу.) Во-вторых, указаны ортогональные направления, по которым идиомы и реализации шаблонов могут отличаться друг от друга. Для разработчиков шаблонов и программистов, занимающихся реализацией шаблонов проектирования, эта информация крайне важна, однако вы вряд ли найдете ее в других источниках. В заключение, читатели могут свободно загрузить исходный код библиотеки шаблонов Loki, описанной в этой книге, и изучить ее содержание. С ее помощью вы можете не только испытать свой компилятор, но и начать собственную разработку. Разумеется, вы можете вполне законно использовать код, написанный Андреем, для своих целей. Я абсолютно уверен, что ему это будет приятно.

Скотт Мейерс (Scott Meyers)

Предисловие Джона Влиссидеса

Что нового можно сказать о языке C++? Оказывается, очень много. Эта книга посвящена слиянию разных способов программирования — обобщенного программирования, шаблонного метапрограммирования, объектно-ориентированного программирования и разработки шаблонов проектирования — в рамках нового подхода. До сих пор эти направления в программировании развивались изолированно друг от друга, и выгоды, полученные от их объединения, лишь начинают получать достойную оценку. Это слияние открывает новые перспективы для языка C++ не только с точки зрения собственно программирования, но для разработки программного обеспечения в целом. Особенно значительно это повлияет на анализ программного обеспечения и его архитектуру.

Обобщенные компоненты, созданные Андреем, поднимают уровень абстракции настолько высоко, что язык C++ приобретает черты языка спецификаций проектирования (design specification language). При этом в отличие от узкоспециализированных языков проектирования язык C++ сохраняет всю свою мощь и выразительность. Андрей продемонстрировал, как программируются концепции проектирования: синглтоны (singletones), инспекторы (visitors), заместители (proxies), абстрактные фабрики (abstract factories) и т.п. Можно даже настраивать готовые компоненты с помощью шаблонных параметров, не расходуя дополнительного машинного времени. Не нужно выбрасывать кучу денег на разработку новых инструментальных средств или изучать тома методологической тарабарщины. Достаточно иметь надежный современный компилятор (и эту книгу).

Разработчики генераторов кода долгие годы обещали обеспечить их совместимость, но теоретические исследования и практический опыт убедили меня, что достичь этой цели невозможно. Остаются нерешенными проблемы полного обхода дерева поиска, генерации недостаточно качественного кода, негибких генераторов, нечитабельности сгенерированного кода и, разумеется, широко известная проблема, которую можно сформулировать так: “Я не могу вставить этот проклятый код в свою программу”. Каждой из этой проблем достаточно, чтобы завести программиста в тупик, а вместе они создают практически непреодолимые препятствия для автоматической генерации кода.

Как было бы хорошо получить все теоретические преимущества автоматической генерации кода — скорость, легкость реализации, сокращенную избыточность, меньшее количество ошибок, одновременно избежав его практических недостатков! Именно это обещает подход, предложенный Андреем. Обобщенные компоненты (generic components) реализуют удачные схемы в виде удобных для использования, поддающихся смешиванию и подходящих для решения задачи шаблонов (mixable-and-matchable templates). Эти шаблоны делают практически то же, что и генераторы кода: создают стереотипные фрагменты кода для дальнейшей обработки с помощью компилятора. Отличие заключается в том, что шаблоны позволяют сделать это, не выходя за рамки языка C++. В результате происходит полная интеграция полученного кода с исходным кодом приложения. При этом остается возможность использовать всю мощь языка, расширяя классы, замещая методы и подгоняя шаблоны под свои требования.

Некоторые из описанных приемов программирования довольно трудно понять, особенно шаблонное метапрограммирование, рассмотренное в главе 3. Однако, освоив его, вы сможете постичь всю теорию обобщенных компонентов, которые практически сами себя создают. Эти компоненты описаны в последующих главах. Я думаю, что шаблонное метапрограммирование, изложенное в главе 3, само по себе достойно отдельной книги. Остальные десять глав освещают способы его применения. Несмотря на то что десять глав — это довольно много, ваши инвестиции окупятся сторицей.

Джон Влиссидес (John Vlissides)

Предисловие

Возможно, вы держите эту книгу в руках, находясь в книжной лавке, раздумывая, покупать ли ее? А может быть вы пришли в библиотеку и решаете, стоит ли тратить время на эту книгу? Знаю, что у вас нет времени, поэтому буду краток. Если вы когда-либо интересовались, как нужно писать программы высокого уровня на языке C++, как справиться с лавиной мелких деталей, загромождающих даже самую простую программу, или как создать компонент, пригодный для повторного использования, который не нужно разрабатывать заново для каждого нового приложения, то эта книга для вас.

Представьте себе такую картину. Вы приходите с производственного совещания, неся в руках груду диаграмм, на которых нацарапаны ваши комментарии. О'кей, говорите вы, тип события, передаваемого от одного объекта к другому, в любом случае не `char`. Это — тип `int`. И вы изменяете одну строку в вашей программе. Интеллектуальный указатель на объект класса `Widget` работает слишком медленно, его следует сделать неконтролируемым. И вы изменяете еще одну строку. Фабрика объектов должна поддерживать новый класс `Gadget`, добавленный соседним отделом. И вы снова изменяете одну строку.

Вы закончили разработку своей программы. Компилируете. Связываете. Готово.

Отлично! Не кажется ли вам, что в этом сценарии что-то не так? Намного правдоподобнее выглядит следующее развитие событий. Вы приходите с производственного совещания взмыленный, поскольку вам предстоит выполнить кучу работы. Вы запускаете глобальный поиск. Удаляете фрагмент. Добавляете фрагмент. Делаете ошибки. Исправляете ошибки... В этом и заключается работа программиста, не так ли? Хотя эта книга и не гарантирует вам исполнение первого сценария, она поможет вам пройти несколько шагов в этом направлении. Здесь предпринята попытка представить язык C++ в новом качестве — языка для разработки архитектуры программного обеспечения.

Традиционно код представляет собой наиболее детализированный и сложный аспект программного обеспечения. Исторически, несмотря на существование языков разных уровней, предназначенных для поддержки методологий проектирования (например, объектной ориентации), между проектом программы и ее кодом лежит пропасть. Это обусловлено тем, что в коде должны быть учтены все мельчайшие детали реализации и множество других побочных моментов. Цель программы в большинстве случаев скрывается за множеством подробностей.

В этой книге представлена коллекция пригодных к повторному использованию проектных решений, называемых *обобщенными компонентами* (generic components), а также способы их разработки. Обобщенные компоненты предоставляют пользователю хорошо известные выгоды, свойственные библиотекам, однако они пригодны для более широкого спектра системных архитектур. Приемы кодирования и реализации сконцентрированы на задачах и моментах, традиционно присущих проектированию, которое обычно *предшествует* собственно кодированию программ. Благодаря своему высокому уровню абстракции обобщенные компоненты позволяют необычайно выразительно, сжато и легко отображать в коде сложные архитектуры.

В обобщенных компонентах воплощены три ветви программирования: проектирование шаблонов, обобщенное программирование и язык C++. Комбинация этих элементов позволила достичь высокого уровня готовности кода к повторному использованию, условно говоря, как в горизонтальном, так и в вертикальном направлениях.

В горизонтальном направлении небольшое количество библиотечного кода позволяет реализовать огромное — в принципе, бесконечное — количество структур и моделей поведения. В вертикальном направлении степень обобщенности этих компонентов делает их пригодными для широчайшего круга программ.

Своим появлением книга обязана проектированию шаблонов, которое позволяет создавать мощные средства решения часто встречающихся задач объектно-ориентированной разработки программ. Проектирование шаблонов — это тщательно отобранные примеры хорошей разработки — рецепты правильных, пригодных к повторному использованию решений задач, возникающих в различных областях. Основной задачей проектирования шаблонов является создание содержательного лексикона (suggestive lexicon) для воплощаемых разработок. Эти шаблоны описывают задачу, ее проверенное временем решение с разными вариантами, а также последствия выбора одного из них. Проектирование шаблонов не связано с конкретными языками программирования. Следуя определенным шаблонам проектирования и комбинируя их друг с другом, компоненты, представленные в этой книге, стремятся охватить как можно более широкий круг конкретных задач.

Обобщенное программирование — это парадигма, в центре которой лежат абстрактные типы, узкий набор функциональных требований и алгоритмы реализации, выраженные в терминах этих требований. Поскольку алгоритмы сами определяют точную и тесную связь с типами, которыми они могут манипулировать, один и тот же алгоритм можно применять для работы с широким спектром типов. Для реализации алгоритмов, приведенных в этой книге, использованы методы обобщенного программирования, позволяющие достичь минимальной специфичности, невероятной лаконичности и эффективности, присущих программам, тщательно разработанным вручную.

В качестве средства реализации в книге используется только язык C++. В этой книге вы не найдете кода, реализующего изящные системы оконного интерфейса, сложных библиотек для сетевого программирования или интеллектуальных механизмов регистрации (logging mechanisms). Вместо этого вы обнаружите массу базовых компонентов, которые облегчают решение как всех описанных выше задач, так и многих других. Для разработки этих компонентов язык C++ крайне необходим. Лежащий в его основе механизм управления памятью, реализованный в языке C, гарантирует быстрое выполнение программ, поддержка полиморфизма позволяет применять приемы объектно-ориентированного программирования, а шаблоны допускают использование невероятных машин, предназначенных для автоматической генерации кода. Шаблоны проходят красной нитью через всю книгу, поскольку они обеспечивают тесное взаимодействие пользователя и библиотеки. Пользователь библиотеки буквально контролирует способ генерации кода, причем этот способ ограничивается самой библиотекой. Предназначение библиотеки обобщенных компонентов — позволять пользователю создавать собственные типы и определять их поведение, а также правильно объединять их с другими обобщенными компонентами. Поскольку при этом используются статические методы, ошибки, связанные со смешиванием и сравнением соответствующих фрагментов, обычно обнаруживаются на этапе компиляции.

Аудитория

Аудитория, которой предназначена эта книга, состоит из двух частей. К первой категории относятся опытные программисты на C++, желающие овладеть наиболее современными методами создания библиотек. В книге представлены новые мощные идиомы языка C++, обладающие удивительными возможностями, некоторые из которых невозможно было себе представить. Эти идиомы окажут неоценимую помощь при создании библиотек

высокого уровня. Для программистов среднего уровня, желающих повысить свою квалификацию, книга также будет полезной, особенно если они проявят определенную настойчивость. Несмотря на то что иногда в книге встречаются довольно сложные фрагменты кода на C++, они всегда сопровождаются подробным комментарием.

Вторая категория состоит из постоянно занятых программистов, которым нужно сделать дело, не тратя лишнего времени на изучение теории. Они могут пропустить наиболее сложные детали реализации и сосредоточить свое внимание на *использовании* предложенной библиотеки. Каждая глава начинается с подробного введения и заканчивается разделом, посвященным часто задаваемым вопросам. Для понимания и использования компонентов эти разделы окажутся весьма полезными. Компоненты можно изучать независимо друг от друга. Они очень мощны и тем не менее безопасны, и, кроме того, их очень легко применять в своих приложениях.

От читателя требуется хорошее знание языка C++ и желание знать его еще лучше. Следует также иметь представление о шаблонах вообще и стандартной библиотеке шаблонов (STL) в частности.

Знание основных шаблонов проектирования (Гамма и др., 1995) желательно, но не обязательно. Идиомы и шаблоны, применяемые в книге, детально описаны. Однако эта книга посвящена другой теме — в ней не делается попыток максимально обобщить шаблоны проектирования. Поскольку они рассматриваются с точки зрения прагматичного создателя библиотеки, даже читатель, интересующийся в основном шаблонами проектирования, найдет для себя много нового, если захочет.

Библиотека Loki

В книге описывается реальная библиотека Loki, написанная на языке C++. Локи (Loki) — это бог остроумия в скандинавской мифологии, и автор надеется, что оригинальность и гибкость этой библиотеки соответствует названию. Все элементы библиотеки находятся в пространстве имен `Loki`. В примерах программ пространство имен не указывается, поскольку это увеличило бы размер кода и затемнило его содержание. Библиотеку Loki можно свободно загрузить с Web-страницы `http://www.awl.com/cseng/titles/0-201-70431-5`.

За исключением части, касающейся потоков, библиотека Loki написана на стандартном языке C++. Увы, это означает, что многие современные компиляторы не смогут работать с ней в полном объеме. Я реализовал и протестировал библиотеку Loki с помощью компиляторов CodeWarrior Pro 6.0 компании Metrowerks и Comeau C++ 4.2.38 (оба компилятора работали под управлением системы Windows). Похоже, что компилятор KAI C++ также не должен иметь с этой библиотекой никаких проблем. Как только поставщики распространят новые, усовершенствованные версии компиляторов, вы сможете эксплуатировать библиотеку Loki полностью.

Код библиотеки Loki, а также примеры, приведенные в книге, используют популярный стандарт кодирования, предложенный Хербом Саттером (Herb Sutter). Я уверен, что вы легко его поймете. Этот стандарт сводится к следующему.

- Классы, функции и перечислимые типы выглядят так: `LikeThis`.
- Переменные и перечислимые значения выглядят так: `likeThis`.
- Переменные-члены выглядят так: `likeThis_`.
- Шаблонные параметры объявляются с ключевым словом `class`, если они представляют собой тип, определяемый пользователем (user-defined type), и с ключевым словом `typename`, если тип является простым (primitive).

Структура книги

Книга состоит из двух основных частей: способы программирования и компоненты. Часть I (главы 1–4) описывает способы программирования на языке C++, используемые в обобщенном программировании и, в частности, для создания обобщенных компонентов. Представлено множество особенностей и способов программирования на языке C++: проектирование, основанное на анализе поведения, частичная специализация шаблонов, списки типов, локальные классы и т.д. Эту часть можно читать последовательно, а затем возвращаться к ней за конкретной информацией.

Часть II организована так же, как и часть I. В ней рассматривается реализация обобщенных компонентов. Здесь нет искусственных примеров. Все описанные компоненты используются в реальных приложениях. Проблемы, ежедневно встающие перед программистами на языке C++, например, интеллектуальные указатели, фабрики объектов и функторы, обсуждаются глубоко и решаются в общем виде. Реализации, приведенные в тексте, ориентируются на основные потребности программистов и предназначены для решения фундаментальных задач. Вместо подробного объяснения, что именно делает тот или иной фрагмент кода, в книге последовательно применяется следующий подход: сначала обсуждается задача, а затем выбирается и реализуется метод ее решения.

Глава 1 посвящена классам стратегий — идиомам языка C++, позволяющим разрабатывать гибкие проектные решения.

В главе 2 обсуждаются основные способы программирования на языке C++, относящиеся к обобщенному программированию.

Списки типов, представляющие собой мощные структуры для манипуляции с типами, реализуются в главе 3.

В главе 4 описывается важный вспомогательный инструмент — механизм распределения памяти для небольших объектов (small-object allocator).

Обобщенные функторы, использующие шаблон проектирования **`Command`**, обсуждаются в главе 5.

В главе 6 описываются синглтоны.

Глава 7 посвящена интеллектуальным указателям.

В главе 8 описываются обобщенные фабрики объектов.

Глава 9 посвящена шаблону проектирования **`Abstract Factory`** и его реализации.

В главе 10 в общем виде реализовано несколько вариантов шаблона проектирования **`Visitor`**.

Механизмы мультиметодов (multimethod engines), представляющие собой решения, ориентированные на использование готовых компонентов, реализованы в главе 11.

Темы, связанные с шаблонами проектирования, охватывают многие ситуации, с которыми постоянно сталкиваются программисты, создающие программы на языке C++. Лично я считаю фабрики объектов (глава 8) краеугольным камнем, лежащим в основе практически всех полиморфных проектных решений. Кроме того, интеллектуальные указатели (глава 7) представляют собой важный компонент многих приложений, созданных с помощью языка C++. Обобщенные функторы (глава 5) чрезвычайно часто встречаются в различных приложениях и позволяют намного упростить сложные проблемы, связанные с проектированием. Другие, более специализированные обобщенные компоненты, такие как `Visitor` (глава 10) или мультиметоды (глава 11), также имеют свою область применения и раздвигают границы языковой поддержки.