Андрей Боровский

C++ и Pascal в Kylix 3

Разработка интернет-приложений и СУБД

- Взаимодействие приложений Kylix 3 с операционной системой Linux
- Интернет-программирование
- Принципы разработки интернет-приложений в Kylix 3
- СУБД InterBase. СУБД MySQL
- Разработка и распространение профессиональных программных продуктов

МАСТЕР_ПРОГРАММ_

Андрей Боровский

C++ и Pascal в Kylix 3

Разработка интернет-приложений и СУБД

Санкт-Петербург
«БХВ-Петербург»
2003

УДК 681.3.06
ББК 32.973.26-018
Б83

Боровский А. Н.

Б83 С++ и Pascal в Kylix 3. Разработка интернет-приложений и СУБД. — СПб.: БХВ-Петербург, 2003. — 544 с.: ил.

ISBN 5-94157-281-6

Книга рассказывает о новейших технологиях программирования на языках C++ и Pascal, реализованных в среде Kylix 3, о поддержке XSL и интерактивной отладке Web-приложений в Kylix IDE. Подробно рассматриваются такие технологии, как WebSnap и WebServices. Описываются особенности низкоуровневого программирования графического интерфейса (взаимодействие с библиотекой Qt library). Изложены такие важные для Kylix-программиста вопросы, как настройка Web-сервисов, создание резидентных Linux-приложений (демонов) и методы решения специфических проблем, возникающих при распространении Kylix-приложений.

Для программистов, имеющих базовые навыки работы в средах Delphi, Borland C++ Builder и предыдущих версиях Kylix

УДК 681.3.06
ББК 32.973.26-018

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Ольга Гурьева*
Компьютерная верстка	*Татьяны Олоновой*
Корректор	*Вера Александрова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 18.03.03.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 43,86.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-281-6

Содержание

Предисловие

Книга, которую вы держите в руках, посвящена практическому использованию среды разработки Kylix 3. Под практическим использованием понимается создание полезных программ (а не только учебных образцов). Можно полагать также, что читатели, собирающиеся использовать среду Kylix для разработки реальных программ, уже знакомы с основами программирования вообще и программированием в визуальных средах в частности и нуждаются в более углубленном изучении возможностей Kylix. Именно на этих предположениях и основана концепция данной книги.

Книга включает в основном методы реализации различных программных технологий в Kylix-приложениях. Некоторые из рассмотренных в книге технологий широко распространены в мире программирования, как, например, технологии SOAP и XSL, другие являются скорее специфичными для продуктов Borland (например, использование XML-документов в качестве локальных баз данных).

Подобный подход определяет и структуру книги. Каждый тематический раздел, посвященный программированию определенного типа приложений в среде Borland Kylix, предваряется введением, рассказывающим об основных принципах соответствующей технологии.

Казалось бы, сама идея визуального программирования нацелена именно на то, чтобы сделать процесс программирования как можно более наглядным. Это относится не только к разработке пользовательского интерфейса приложения. Например, работа с компонентами, обеспечивающими связь с базами данных, организована в Kylix таким образом, что еще на этапе разработки можно *видеть*, как создаваемое приложение будет взаимодействовать с выбранной СУБД (и будет ли оно взаимодействовать с ней вообще). То же самое можно сказать и о технологии разработки Web-приложений WebSnap, которая позволяет просматривать динамические страницы, создаваемые приложением, не покидая интегрированной среды разработки. Ко всему этому следует добавить прекрасную интерактивную справочную систему и объемистые руководства разработчика, поставляемые вместе с Kylix как в бумажном, так и в электронном формате.

И тем не менее мы убеждены, что книги о Kylix необходимы. Сколь бы подробна ни была справочная система, она не может охватить всего. Посе-

тите любой сайт, посвященный программированию для Delphi/Kylix, и вы найдете десятки статей, дополняющих и разъясняющих фирменную документацию. Зайдите на один из множества форумов по Delphi/Kylix-программированию, и вы увидите вопросы, на которые фирменная документация не дает развернутых ответов.

При написании книги автор старался (насколько это возможно) не дублировать справочную систему. Поэтому книга не содержит подробного перечисления и описания всех компонентов Kylix (учитывая общее их количество, это было бы не только нецелесообразно, но и трудноосуществимо). В соответствии с концепцией книги и предполагаемым уровнем подготовки читателя в ней раскрыты основные принципы реализации различных современных программных технологий в Borland Kylix 3, а также принципы разработки приложений, использующих эти технологии (в связи с чем, книга содержит немало практических пошаговых инструкций и примеров текстов готовых программ). Освоив эти принципы, читатель без труда сможет ориентироваться в соответствующих технологиях, прибегая, где это нужно, к услугам встроенной справочной системы.

Кроме перечисленного выше в книге затронута и специфика программирования в ОС Linux, выходящая за пределы Kylix-программирования, как, например, особенности файловой системы и межпроцессного взаимодействия в ОС Linux, настройка Web-серверов и серверов баз данных для работы с Kylix-приложениями, создание дистрибутивов приложений для ОС Linux и т. п.

В общем автор стремился максимально выйти за рамки тех сведений, которые можно получить с помощью клавиши <F1>, а потому книга окажется полезной для широкого круга программистов средней и высокой квалификации.

Введение

Эта книга, как следует из ее названия, посвящена программированию в среде разработки Kylix 3 на языках C++ и Delphi Language (так теперь называется версия Object Pascal).

Что такое Kylix 3?

Borland Kylix 3 представляет собой интегрированную среду разработки, предназначенную для быстрой визуальной разработки приложений на языках C++ и Delphi Language, способных выполняться отдельно от среды. При этом основной упор сделан на быстрой разработке графических интерфейсов (при помощи богатейшего набора графических компонентов, основанного на библиотеке Qt library), разработке Web-приложений и приложений для работы с базами данных. Разработчики Kylix не только стремятся к внедрению в свои продукты новейших технологий, но и делают все возможное для того, чтобы облегчить пользователям освоение новых методов программирования. Для ускорения и упрощения процесса разработки приложений в среде Borland Kylix введен целый ряд специальных функций и возможностей, начиная с первоклассного редактора графического интерфейса и "интеллектуального" редактора исходного текста, позволяющего ускорить ввод текста программы, и заканчивая такими средствами, как интерактивный отладчик приложений для Web-сервера или модель визуальной разработки приложений баз данных. В рамках этой модели взаимодействие с сервером баз данных может выполняться еще на этапе разработки приложения, что делает этот процесс более наглядным и позволяет обнаруживать и устранять на данном этапе некоторые ошибки, проявляющиеся обычно только во время выполнения программы.

Среди достоинств последних версий Kylix нельзя не отметить появление средств разработки, отражающих передовые тенденции в сетевом программировании, в частности, широкого спектра инструментов, которые предназначены для работы с XML-документами и Web-службами. Уникальные решения, предоставляемые Borland, позволяют интегрировать поддержку технологий XML и XSL в приложения самых разных типов, в том числе в локальные и распределенные приложения баз данных.

Появление в новой версии Kylix интегрированной среды разработки на языке C++ позволяет не только использовать мощь и гибкость этого языка, но и напрямую обращаться к различным интерфейсам программирования ОС Linux.

Ко всему перечисленному выше следует добавить подробную и удобную в использовании интерактивную справочную систему, которая позволяет получить доступ не только к справке по функциям Kylix и элементам реализованных в этой среде языков программирования, но и к встроенной в операционной системе Linux справочной системе man.

Средства разработки Kylix совместимы с аналогичными продуктами Borland для платформы Win32, что делает возможным перенос проектов между платформами ОС Linux и Windows. Вместе с тем Borland Kylix обладает рядом существенных отличий, связанных прежде всего со спецификой ОС Linux и средствами взаимодействия с Интернетом и межпроцессного взаимодействия, реализованными в этой операционной системе. Неизбежные различия возникают в связи с принципиальной разницей интерфейсов прикладного программирования (API) ОС Linux и Windows, а также вследствие некоторых особенностей, присущих вариантам языков C и C++, традиционно используемых в ОС Linux.

Длительная история разработки компанией Borland визуальных сред программирования для ОС Windows и Linux позволяет сделать однозначные выводы о стратегических направлениях развития этих средств. Очевидно, что основной задачей Borland является создание средств быстрой разработки приложений для корпоративного сектора, а также приложений для ведения электронного бизнеса в Глобальной сети.

По сравнению с предыдущими версиями в Kylix 3 введен целый ряд дополнений. О главном новшестве — компиляторе языка C++ уже говорилось. Среди других достоинств версии 3 заслуживают внимания следующие:

- поддержка инструкций Pentium 4 во встроенном ассемблере;
- предварительная компиляция заголовочных файлов C++ (precompiled headers) для ускорения компиляции приложений;
- встроенный интерактивный отладчик Web-приложений;
- редактор связей между объектами (окно **Diagram**) позволяет отображать и редактировать эти связи через свойства;
- поддержка работы с XSL-шаблонами (компонент XSLPageProducer);
- новые драйверы набора компонентов DBExpress.

Зачем нужен Kylix 3?1

Подобный вопрос приходится слышать нередко, причем те, кто его задает, иногда напоминают, что для ОС Linux уже существуют бесплатные реализации языка Pascal. К ним относится интегрированная среда Lazarus, стремящаяся, как говорят ее разработчики, стать максимально похожей и полностью

совместимой, по крайней мере, для исходных текстов, со средой Delphi (но пока этот проект весьма далек от намеченной цели). Те, кто задают подобные вопросы, забывают, что Kylix — это не просто "Object Pascal для Linux", что, кстати говоря, вообще перестало быть актуальным с введением в Kylix 3 среды программирования C++. Borland Kylix — это прежде всего средство разработки наиболее распространенных на сегодняшний день типов приложений, использующих последние технологии в таких областях, как интернет-программирование и работа с базами данных. Осмелимся утверждать, что сегодня в мире ОС Linux нет другого средства разработки, позволяющего решать те же задачи с той же быстротой и удобством. Если до недавнего времени программистов, работающих под ОС Linux, могла отпугивать необходимость использовать при работе с Kylix нелюбимый многими язык Pascal, то теперь с появлением интегрированного пакета, содержащего средства разработки как на языке Delphi Language, так и на языке C++, эта проблема также отпадает. Разумеется, Borland Kylix не претендует (и никогда не претендовал) на роль средства системного программирования. Этот продукт предназначен для быстрой разработки прикладных программ. Если Kylix Open Edition предоставляет в распоряжение пользователя удобное средство визуальной разработки и широкий набор графических компонентов (не говоря уже о компиляторах языков Delphi Language и C++ и удобнейшем редакторе исходного текста), то Kylix Enterprise вполне может служить основой для разработки корпоративных решений в рамках предприятия.

Что вы найдете в этой книге?

Эта книга посвящена практическому использованию различных технологий программирования, реализованных в Kylix 3. При этом основной акцент сделан на новых компонентах и технологиях, появившихся в версиях 2 и 3 Kylix. Перед тем, как описать содержимое книги, будет весьма уместно перечислить то, чего читатель в ней не найдет.

В книге не излагаются основы языков программирования Delphi Language и C++, а также концепции объектно-ориентированного программирования и работы в интегрированной среде разработки. Отсутствует описание простейших методов визуального программирования и построения стандартных графических интерфейсов приложений. Если читатель нуждается в подобных сведениях, можно рекомендовать предварительно прочесть одну из книг, являющихся введением в программирование для Delphi/Kylix. Поскольку многие из тех, кто выбирает Kylix в качестве средства разработки, уже знакомы с этими вводными вопросами (например, по опыту программирования в среде Delphi), представляется предпочтительным сосредоточить внимание на более сложных (и не столь широко освещенных) аспектах программирования в Kylix для ОС Linux.

Три первые части книги соответствуют трем основным сферам применения Borland Kylix. *Часть I* представляет собой введение в прикладное программирование для ОС Linux. Эта часть предназначена для разработчиков, незнакомых с основами программирования в указанной операционной системе, а также для Linux-программистов, стремящихся "адаптироваться" к среде программирования Borland Kylix. *Глава 1* данной части включает сведения о разработке консольных приложений и разделяемых библиотек ОС Linux с помощью Borland Kylix, взаимодействии Kylix-приложений с интерфейсами прикладного программирования ОС Linux, особенностях работы с файловой системой этой ОС (включая виртуальные файловые системы), механизмах управления процессами и межпроцессного взаимодействия. *Глава 2 части I* посвящена тонкостям программирования графического интерфейса Kylix-приложений (рассматриваются примеры на языках Delphi Language и C++) и описанию взаимодействия графических компонентов Kylix с графической подсистемой ОС Linux. В этой главе читатель найдет также примеры методов запуска графических приложений из Kylix-приложений, а также использования в Kylix-приложениях альтернативных графических интерфейсов.

Изложение вышеуказанного материала сопровождается необходимыми пояснениями, касающимися принципов работы ОС Linux и, естественно, примерами, написанными с использованием как языка Delphi Language, так и языка C++. При этом определенное внимание уделено особенностям реализации языка C++ в Kylix и соотношению этих особенностей с традициями программирования на языках C и C++, принятыми в ОС Linux.

Часть II книги посвящена интернет-программированию. Поскольку часть II посвящена разработке приложений для Web-серверов, в первой главе этой части *(глава 3)* дается краткое описание сервера Apache и других Web-серверов, распространенных на платформе ОС Linux, излагаются принципы разработки модулей для сервера Apache и CGI-приложений. *Глава 4* является вводно-обзорной главой, кратко описывающей методы разработки сетевых приложений, более подробно изложенных в последующих главах. В этой главе приводятся примеры создания простейших интернет-приложений "с нуля" без использования специфических компонентов Kylix. Основная цель этих примеров заключается в том, чтобы помочь читателю освоить принципы работы сетевой подсистемы ОС Linux (хотя в определенных ситуациях эти примеры, демонстрирующие возможности разработки "легких" сетевых приложений с помощью Kylix, могут быть полезны и сами по себе). Далее следует краткое описание специальных средств разработки сетевых приложений различных типов, а также средств отладки этих приложений.

Глава 5 посвящена разработке независимых интернет-приложений на основе набора компонентов Internet Direct (Indy). При этом подробно рассматривается многопоточная модель Indy, принципы реализации TCP и UDP-

протоколов с помощью этого набора компонентов, приводятся примеры приложений, осуществляющих с помощью Indy работу с популярными интернет-протоколами HTTP и FTP.

Поскольку программирование интернет-приложений в Kylix тесно связано с новейшими интернет-технологиями, автор счел необходимым посвятить отдельную *главу 6* описанию языка XML и его производных. Также в этой главе рассматриваются инструменты Borland Kylix, предназначенные для работы с языком XML.

Три следующие главы 7, 8 и 9 части III посвящены "трем китам" Web-программирования в Borland Kylix — технологиям WebBroker (быстрая разработка Web-приложений и динамическая генерация контента), WebSnap (автоматизация распространенных задач Web-программирования и технология сценариев на стороне сервера) и WebServices (построение распределенных приложений на основе протокола SOAP). В рамках каждой главы дается описание основных компонентов Kylix, реализующих соответствующую технологию, и рассматриваются примеры использования этих компонентов в приложениях для Web-сервера, написанных на языках C++ и Delphi Language.

При описании технологии WebBroker подробно рассматривается механизм диспетчеризации запросов, формирования HTTP-ответов и использования компонентов-генераторов контента. В качестве примера приводится Web-приложение, использующее технологию cookies для идентификации пользователей.

В главе 8, посвященной технологии WebSnap, приводятся примеры написания сценариев на стороне сервера на языке JavaScript, описываются механизмы работы компонентов-адаптеров, команд и полей компонентов-адаптеров различных типов. Эта глава включает также примеры использования других важных компонентов технологии WebSnap и приемов визуального программирования Web-приложений в рамках данной технологии.

Глава 9, посвященная технологии WebServices, начинается с краткого описания протокола SOAP и языка WSDL. Объяснение принципов работы компонентов WebServices и других инструментов Kylix, предназначенных для работы с этой технологией, сопровождается примерами распределенных SOAP-систем, написанных на языках C++ и Delphi Language.

Завершает *часть II* книги *глава 10*, посвященная описанию программирования распределенных приложений на основе технологии CORBA с использованием брокера запросов VisiBroker.

Часть III книги посвящена разработке локальных и распределенных приложений баз данных с помощью Borland Kylix. При этом, как и в предыдущих частях, примеры предваряются необходимыми теоретическими сведениями.

В главе 11 излагаются основные принципы реляционной модели баз данных и дается описание общей архитектуры приложений баз данных в Borland Kylix.

Главы 12 и *13* посвящены описанию настройки и работы с двумя распространенными в мире ОС Linux системами управления базами данных (СУБД) — InterBase и MySQL.

Поскольку компоненты dbExpress, являющиеся основой механизма взаимодействия Kylix-приложений с базами данных, используют язык запросов SQL, описание компонентов dbExpress, которому посвящена *глава 14*, предваряется кратким описанием языка SQL.

В главе 15, посвященной локальным приложениям баз данных, рассматривается работа с клиентскими наборами данных, компонентами-провайдерами и элементами пользовательского интерфейса приложений баз данных. На примерах нескольких баз данных описываются такие операции, как поиск и фильтрация записей, создание индексов и закладок, хранене в базах данных изображений и т. п. Также в этой главе указаны средства и методы хранения данных в XML-документах и механизмы взаимодействия между приложениями, использующими в качестве хранилища данных XML-документы и классические приложения баз данных.

Глава 16, посвященная описанию распределенных приложений баз данных, начинается с модели "портфеля", которая обеспечивает возможность работы клиентских программ распределенных систем баз данных в автономном режиме. Далее следует представление многоуровневой модели распределенных приложений баз данных, реализованной на основе протокола SOAP. Заключает главу 16 описание методов разработки Web-приложений баз данных, в рамках которого рассматривается применение в приложениях баз данных технологий WebBroker и WebSnap.

Заключительная, *часть IV* книги содержит дополнительные главы, посвященные проблемам, возникающим перед разработчиками готовых программных продуктов.

В главе 17 рассматриваются вопросы разработки компонентов для среды Borland Kylix. Излишне напоминать, что в настоящее время компоненты для интегрированных сред разработки часто распространяются как коммерческие программные продукты.

Глава 18 посвящена вопросам сопровождения и распространения Kylix-приложений. В этой главе читатель найдет разделы, посвященные созданию справочной системы для приложений Kylix, а также различным методам распространения готовых приложений.

Глава 19 — единственная глава книги, непосредственно не связанная с программированием в Borland Kylix. В этой главе рассматриваются особенности различных архитектур приложений электронного бизнеса, на которые ориентировались разработчики Kylix.

Для кого предназначена эта книга?

С учетом уровня изложения материала и концепции книги от читателя требуется наличие определенных предварительных знаний. Примеры программ в книге приводятся на языках программирования C++ и Delphi Language, так что предполагается владение хотя бы одним из этих языков. Предполагаются также знакомство с концепциями объектно-ориентированного программирования и навыки разработки приложений в визуальной среде.

Таким образом, можно представить себе потенциального читателя книги как программиста, имеющего опыт работы с Delphi или C++ Builder и желающего освоить разработку приложений на платформе ОС Linux, либо как Linux-разработчика, стремящегося освоить технологии программирования интернет-приложений и приложений баз данных, предоставляемые средой Borland Kylix.

Впрочем, можно надеяться, что книга окажется полезной для всех, кто интересуется программированием в средах Borland или программированием для ОС Linux.

Часть I

Взаимодействие приложений Kylix 3 с операционной системой

Глава 1

Kylix-приложения и прикладные интерфейсы Linux

В этой главе мы рассмотрим основные аспекты взаимодействия приложений, созданных в среде Kylix 3, с ОС Linux. Данная глава посвящена особенностям разработки консольных приложений, работе с файловой системой Linux, взаимодействию приложений, написанных на языке C++ и Delphi Language, с системными библиотеками, механизмам взаимодействия между процессами и особенностям программирования на языках C/C++ в ОС Linux.

Особенности языка C++ в Borland Kylix

Особенности реализации языка C++ в среде разработки Borland Kylix 3 определяются, с одной стороны, традициями компании Borland, а с другой стороны, спецификой ОС Linux. Если у вас есть опыт работы с Borland C++ Builder, новая среда для Linux не должна показаться вам незнакомой, ведь основной задачей Borland и было создание среды разработки, облегчающей перенос C++ проектов из среды Windows на платформу Linux. Тем не менее, некоторые особенности ОС Linux потребовали от разработчиков введения новых возможностей. Далее мы рассмотрим некоторые характерные черты Kylix C++, отличающие этот язык от варианта C++ Builder и от знакомого Linux-программистам GNU C++.

Первая характерная особенность Borland C++ Builder связана с директивами компилятора. В этом варианте традиционно используется ряд специфических директив `#pragma`. Собственно говоря, смысл директив `#pragma` и заключается в том, что эти директивы позволяют компилятору устанавливать опции, не конфликтующие с опциями других компиляторов. Если компилятор встречает в тексте программы незнакомую директиву `#pragma`, он ее просто игнорирует. Использование директив `#pragma` позволяет перенести в текст программы многие опции компилятора Kylix, обычно сохраняемые в специальных файлах проектов. Таким образом, с помощью

директив `#pragma` можно создавать программы, предназначенные для компиляции разными компиляторами C++.

Полный список директив `#pragma` приводится в справочной системе Kylix C++ IDE. Мы же в качестве примера рассмотрим здесь лишь две директивы — `#pragma link` и `#pragma hdrstop`. Директива `#pragma link` позволяет указывать список модулей, связываемых с программой на этапе компоновки, например, для того чтобы связать приложение с библиотекой libGL.so указываем:

```
#pragma link "/usr/X11R6/lib/libGL.so"
```

Как видим, `#pragma link` позволяет указать не только имя модуля, но и путь к нему, чем эта опция выгодно отличается от директив связывания разделяемых библиотек в Delphi Language IDE. С помощью директивы `#pragma link` можно связывать с приложением и объектные файлы:

`#pragma link` "objfile.o" (в Linux объектные файлы имеют расширение .O).

Впрочем, в C++ IDE связывать внешние модули с проектом можно при помощи команд меню **Project | Add to Project**.

Директива `#pragma hdrstop` находится в тесной связи с другой особенностью Borland C++ — предварительной компиляцией заголовков. Предварительная компиляция заголовков предназначена для того, чтобы хотя бы отчасти преодолеть один из основных недостатков компиляторов C++ — низкую скорость сборки приложений.

Существенно более низкая скорость компиляции проектов на C++ по сравнению с программами, написанными на стандартном C, связана отчасти с гораздо более сложным анализом заголовочных файлов, содержащих объявления классов C++, который приходится выполнять компилятору. Для решения этой проблемы разработчики Borland и предложили использовать предварительную компиляцию заголовочных файлов (под компиляцией в данном случае подразумевается построение таблиц символов (symbol tables)). Выигрыш в скорости основан на том, что заголовочные файлы, как правило, — самая консервативная часть проекта, а заголовки стандартных библиотек разработчиками программ вообще не модифицируются. Таким образом, выполнив компиляцию заголовков один раз, можно существенно ускорить процесс сборки приложений.

Практически эта технология реализована следующим образом. Для каждого набора директив `#include`, встречающегося в исходных файлах проекта, создается отображение (image), содержащее скомпилированные заголовки в той последовательности, в которой они включены в исходный файл. Все отображения, созданные для данного проекта, сохраняются в специальном файле. Совместное использование одного отображения скомпилированных заголовков несколькими модулями возможно только в том случае, если заголовки включены в исходные файлы в одной и той же последовательности.

Поскольку секции исходных файлов, содержащие директивы `#include`, как правило, совпадают лишь частично, для совместного использования отображений скомпилированных заголовков и введена директива `#pragma hdrstop`. Если указанная директива встречается в исходном файле, предварительная компиляция выполняется только для заголовочных файлов, включенных перед этой директивой. Благодаря директиве `#pragma hdrstop` исходные файлы

```
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#pragma hdrstop
#include "unit1.h"
...
```

и

```
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#pragma hdrstop
#include "unit2.h"
...
```

могут использовать одно и то же отображение заголовков.

Включение/отключение режима предварительной компиляции заголовков управляется опциями раздела **Precompiled Headers** вкладки **Compiler** окна **Project Options** (по умолчанию режим включен). На этой вкладке можно также указать имя файла, в котором будут храниться скомпилированные заголовки для данного проекта (в противном случае компилятор сам выберет это имя). На этой же вкладке можно установить и значение **Stop After**, имеющее тот же смысл, что и директива `#pragma hdrstop`.

При желании вы можете включить в число предварительно компилируемых заголовков и те заголовочные файлы, что были созданы в рамках вашего проекта, если, конечно, вы уверены в их неизменности. Не следует включать в число предварительно компилируемых заголовочных файлов те, на которые ссылаются другие предварительно компилируемые заголовки.

Еще одна особенность Borland C++ связана с объектными файлами. В отличие от Delphi IDE, компилятор C++ генерирует объектные файлы для всех модулей проекта. Эти файлы полностью соответствуют стандарту объектных файлов компилятора GCC и экспортируют таблицы имен, которые можно просматривать с помощью стандартной Linux-команды `nm`.

Листинг 1.1. Фрагмент вывода команды `nm` для объектного файла модуля Kylix C++

```
00000000 W @TForm1@$bctr$qqrp18Classes@TComponent
00000000 W @TForm1@$bdtr$qqrv
00000000 W @TForm1@Button1Click$qqrp14System@TObject
00000000 W @TForm1@Button2Click$qqrp14System@TObject
00000000 W @TForm1@FormClose$qqrp14System@TObjectr19Qforms@TCloseAction
```

Таким образом, существует принципиальная возможность разделения объектных файлов между компиляторами Kylix и GCC. Однако на этом пути возникают существенные ограничения. Дело в том, что форматы хранения классов в объектных файлах Kylix и GCC различаются. Классы Borland и GCC различаются не только форматом вызова методов (в Kylix это fastcall, а в GCC — cdecl), но и схемой изменения имен (names mangling). В листинге 1.1 были приведены примеры экспортируемых имен методов, созданных компилятором Kylix. Сравним их с именами методов, сгенерированных компилятором GCC (листинг 1.2).

Листинг 1.2. Фрагмент вывода команды `nm` для объектного файла, созданного с компилятором GCC

```
0000002c T _._7QtGLDoc
00000000 T __7QtGLDoc
000000a4 T isModified__C7QtGLDoc
00000088 T load__7QtGLDocRC7QString
00000068 T newDoc__7QtGLDoc
0000007c T saveAs__7QtGLDocRC7QString
```

Как говорится, разница очевидна. Механизмы выделения и высвобождения памяти для экземпляров классов в Kylix и GCC также различаются. Одним из следствий различия форматов классов является то, что в Borland Kylix вы не сможете непосредственно использовать библиотеки Linux (как статические, так и динамические), экспортирующие классы C++. Разделение внешних модулей между GCC и Kylix возможно только в том случае, если эти модули написаны на стандартном языке C. GCC по умолчанию генерирует объектный код, основанный на правилах C, для всех файлов с расширением .c, а в Kylix для создания C-модуля следует выбрать пункт **C File** страницы **New** диалогового окна **New Items**.

Для программистов, привыкших к GNU C++, следует отметить такую особенность Borland C++, как свойства (properties). Свойства инкапсулируют поля классов, для чего в традиционном C++ приходится создавать многочисленные пары процедура/функция. Borland C++ позволяет определять свойства самых разных типов, включая такие, как указатель на экземпляр класса или массив записей.

Как и GCC, компилятор Borland позволяет использовать директивы встраивания функций в программах, написанных на языке C. Только в случае C в Borland Kylix следует использовать директиву `__inline`.

Как и в Borland C++ Builder, в Kylix C++ существует несколько встроенных макросов, предназначенных для условной компиляции кода. Прежде всего, это макрос `__BCPLUSPLUS__`, означающий, что программа компилируется в среде Borland C++. Кроме того, компилятор поддерживает стандартный макрос `__cplusplus`. Если вы пишете программу на стандартном C, то для выделения фрагмента кода, предназначенного для компилятора Borland, можно использовать макрос `__BORLANDC__` (значение этого макроса указывает номер версии компилятора). Для выделения платформо-зависимых фрагментов кода можно использовать макросы `#ifdef WINDOWS` и `#ifdef __linux__`.

Как уже отмечалось, среда Kylix 3 позволяет создавать программы и на C++, и на Delphi Language, поэтому в данной книге исходные тексты программ приводятся на этих языках. Дублировать текст каждой программы "переводом" на другой язык программирования было бы нерационально, тем более что большинство читателей этой книги, надеемся, владеет и C++, и Pascal. Тем не менее специально для тех, кто незнаком с одним из этих языков, приводится несколько замечаний, призванных облегчить "перевод" листингов. В табл. 1.1 приведено сравнительное описание типов в C++ и Delphi Language.

Таблица 1.1. *Сравнение типов C++ и Delphi Language*

Тип C++	Тип Delphi Language	Описание типа	Размер, байт
		Целые типы	
`Char`	`ShortInt`, `Char`	Знаковый 8-битный или символ	1
`unsigned char`	`Byte`	Беззнаковый 8-битный	1
`short`	`SmallInt`	Знаковый 16-битный	2
`unsigned short`	`Word`	Беззнаковый 16-битный	2
`int`, `long`	`Integer`, `LongInt`	Знаковый 32-битный	4
`unsigned long`, `unsigned int`	`LongWord`, `Cardinal`	Беззнаковый 32-битный	4
`long long`	`Int64`	Знаковый 64-битный	8
`unsigned long long`	—	Беззнаковый 64-битный	8

Таблица 1.1 *(окончание)*

Тип C++	Тип Delphi Language	Описание типа	Размер, байт
	Типы чисел с плавающей точкой		
`float`	Single	7—8 значащих цифр	4
`double`	`Double`	15—16 значащих цифр	8
`long double`	`Extended`	19—20 значащих цифр	10
	Логические типы		
bool	Boolean	Логический тип (`True`/`False`)	1
	Типы-указатели		
`void *`	`Pointer`	Нетипизированный указатель	4
`char *`	`PChar`	Указатель на тип `char`, массив типа `char`, строка, заканчивающаяся нулем	4, длина строки произвольная
`some_type *`	`^Some_Type`	Указатель на переменную типа `Type`	4

Отметим еще несколько особенностей C++ и Delphi Language. Для обращения к полям структуры (записи), заданной указателем, в C++ используется оператор `->`. Следующие конструкции эквивалентны:

C++:

```
some_struct->some_field
```

Стандартный Pascal:

```
some_struct^.some_field
```

Delphi Language:

```
some_struct.some_field
```

Программистам, пишущим на языке Pascal, следует учитывать еще одну особенность языков C/C++. В этих языках типы `указатель на тип` и `массив типа` формально взаимозаменяемы. Что имеется в виду в данном конкретном случае, следует смотреть по контексту. Например, в объявлении

```
void some_func(int * retval);
```

переменная `retval` скорее всего является указателем на переменную типа `int`, и в Delphi Language эту декларацию можно перевести как

```
procedure some_func(var retval : Integer); cdecl;
```

а вот в объявлении

```
int ** ppint;
```

переменная `ppint` может интерпретироваться и как массив указателей на переменную типа `int`, и как указатель на массив переменных типа `int`, и как указатель на указатель на переменную типа `int`. Для того чтобы понять, какой именно смысл имеет переменная, придется разбираться в логике работы модуля, содержащего это объявление. Стоит отметить, что неверная интерпретация смысла указателей является, пожалуй, самым частым источником ошибок при программировании. Учтите также, что в C/C++ индексация массивов всегда начинается с нуля, так что объявление переменной

```
int a[ARRAY_SIZE];
```

следует переводить на Pascal как

```
a : array[0..ARRAY_SIZE-1] of Integer;
```

Еще одна особенность C/C++ заключается в том, что в операциях сравнения могут использоваться не только логические, но и целые типы. При этом логическому значению `False` соответствует целое значение 0, а другие значения целочисленной переменной интерпретируются как `True`. Так что C-конструкцию

```
if(non_boolean_val) ...
```

можно перевести на язык Pascal, как

```
if non_boolean_val <> 0 then ...
```

Консольные приложения на языках C++ и Delphi Language

В этом разделе мы познакомимся с особенностями разработки консольных приложений (приложения для текстового терминала) для ОС Linux, функциями библиотеки glibc и методами обработки ошибок, возникающих при вызове этих функций. Разработчики, переходящие с платформы Windows на платформу Linux, должны иметь в виду, что в ОС Linux консольные приложения играют большую роль.

В ОС Linux графическая подсистема отделена от остальной операционной системы, и в принципе эта 32-разрядная многозадачная ОС может использоваться вообще без графического интерфейса. У такого подхода есть свои преимущества: при решении ресурсоемких задач вся мощность процессора может быть брошена на выполнение полезной работы, а не затрачиваться на перерисовку окон. Кроме того, приложениям с текстовым интерфейсом присуща бóльшая компактность, стабильность и меньшая зависимость от внешних библиотек, которые могут быть (а могут и не быть) установлены в конкретной системе. Сам интерфейс командной строки ОС Linux предос-

тавляет пользователю гораздо большие возможности, нежели командная строка DOS/Windows, и многие задачи в ОС Linux гораздо удобнее решать именно из командной строки. Многие важные типы приложений Linux, например приложения-демоны или внешние утилиты, предназначенные для использования в сценариях оболочки, должны быть консольными. Вот почему, изучая программирование для ОС Linux, следует уделить особое внимание вопросам создания консольных приложений.

В этом разделе будет рассмотрено несколько простых консольных программ — для Kylix C++ и Kylix Delphi. Мы также коснемся некоторых проблем, которые возникают в связи с программированием консольных приложений, но имеют значение и при разработке приложений других типов. Так что, даже если не стоит задача писать консольные программы, все равно полезно будет прочесть этот раздел.

При создании проекта консольного приложения C/C++ среда Kylix выводит дополнительное диалоговое окно (рис. 1.1). В этом диалоговом окне можно указать язык консольного приложения (C или C++) и необходимость поддерживать потоки в приложении. Если в качестве языка приложения вы выбрали C++, то с помощью специального флажка вы можете указать также, будет ли приложение использовать классы CLX или нет.

Полезной опцией является компонент ввода **Specify Project Source**, с помощью которого в проект можно импортировать уже готовые тексты C/C++ приложений.

Рассмотрим простое консольное приложение, написанное в Borland Kylix на языке C++ (листинг 1.3).

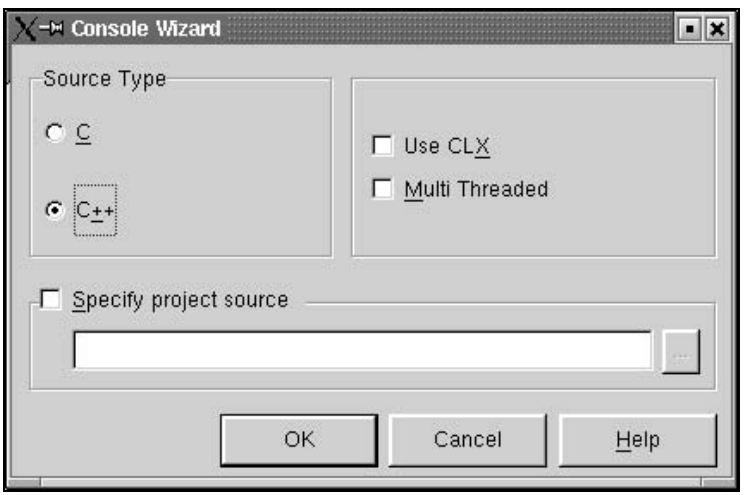

Рис. 1.1. Окно **Console Wizard**

Листинг 1.3. Консольное C++ приложение для Kylix 3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#pragma hdrstop

#pragma argsused

enum {ERROR = -1, SUCCESS, FAIL};

#define BUFSIZE 0x1000
static char buf[2][BUFSIZE];

int fcompare(const char *fname1, const char *fname2)
{
  FILE *f1, *f2;
  int retval = SUCCESS;
  if ((f1 = fopen(fname1, "rb")) == NULL)
  return ERROR;
  if ((f2 = fopen(fname2, "rb")) != NULL)
  {
    size_t size1, size2;
    do
    {
       size1 = fread(buf[0], 1, BUFSIZE, f1);
       size2 = fread(buf[1], 1, BUFSIZE, f2);
       if (0 == (size1 | size2)) break;
       if ((size1 != size2) || memcmp(buf[0], buf[1], size1))
       {
          retval = FAIL;
          break;
       }
    } while (size1 && size2);
    fclose(f2);
  }
  else  retval = ERROR;
```

```
  fclose(f1);
  return retval;
}

int main(int argc, char *argv[])
{
  if (argc != 3)
  {
    puts("Использование: %s file1 file2\n", argv[0]);
    return 0;
  }
  int result = fcompare(argv[1], argv[2]);
  printf("Файлы %s и %s %s\n", argv[1], argv[2],
           result == SUCCESS ? "совпадают" : "не совпадают");
  return result == SUCCESS ? 0 : 1;
}
```

Эта программа сравнивает два файла, переданных ей в качестве параметров при запуске. Программа не только выводит сообщение о совпадении или несовпадении файлов, но и возвращает коды выхода 0, если файлы совпали, и 1 — в противном случае, так что ее можно использовать и в командной строке, и в файлах сценариев.

Как видим, простейшая программа для Kylix C++ ничем не отличается от программ для других компиляторов. Для создания консольной программы на C/C++ в Kylix не нужно указывать специальных директив компилятора, как в случае с Delphi Language. Текст такой программы можно распространять без файла проекта (при открытии файла с программой в Kylix 3 C++ IDE файл проекта будет сгенерирован автоматически).

В случае программы, указанной выше, нам не следует беспокоиться о подключении системных библиотек, так как все функции, используемые в этой программе, определены в библиотеке `glibc`, которая автоматически подключается компоновщиком Kylix.

В случае если программе необходимы функции, экспортируемые другими библиотеками ОС Linux, эти библиотеки придется указывать явно либо в настройках проекта, либо при помощи директивы `#pragma link`.

Большая часть библиотек ОС Linux существует в двух вариантах: динамическом и статическом. Динамические библиотеки Linux подобны библиотекам DLL Windows. Эти библиотеки не включаются в код приложения и позволяют нескольким процессам разделять один и тот же файл. В ОС Linux динамические библиотеки хранятся в файлах с расширением .so. Отличительной особенностью динамических библиотек Linux является наличие у одной

библиотеки нескольких имен (подробнее этот вопрос будет рассмотрен в разделе, посвященном созданию разделяемых библиотек).

Статические библиотеки содержат объектный код, который статически связывается с программой во время ее компоновки. В ОС Linux файлы статических библиотек имеют расширение A. Фактически статические библиотеки можно рассматривать как совокупность объектных файлов. Компоновщики различных компиляторов находят эти библиотеки и включают их по мере необходимости в образ (image) исполнимой программы.

В отличие от Kylix Delphi, Kylix C++ позволяет использовать в программах не только динамические, но и статические библиотеки, так что перед разработчиком возникает вопрос: какой тип библиотек выбрать?

У каждого подхода есть свои достоинства и недостатки. Использование динамических библиотек позволяет уменьшить размер исполнимых модулей. Экономия особенно ощутима, когда несколько программ используют одну и ту же библиотеку. Специфическим неудобством динамических библиотек в ОС Linux (по сравнению с Windows) является то, что в разных дистрибутивах ОС Linux одни и те же библиотеки могут располагаться в разных каталогах и иметь разные имена. Последнее обстоятельство связано с тем, что в ОС Linux номер версии библиотеки является частью ее имени. Кроме того, многие библиотеки Linux не имеют обратной совместимости с прежними версиями. Для открытого программного обеспечения, распространяемого в виде исходных текстов и компилируемого специально для каждого дистрибутива, все это не представляет проблемы, а вот при распространении коммерческих приложений можно столкнуться с некоторыми трудностями. С одним из методов решения этой проблемы мы познакомимся в разделе "Разделяемые библиотеки и объектные файлы" этой главы.

При использовании статических библиотек размер файла программы может существенно увеличиться, но статические библиотеки являются частью программы и уменьшают ее зависимость от внешних модулей.

Говоря о библиотеке `glibc`, следует отметить особенность получения справочных данных по ее функциям. Справочная система Kylix не содержит сведений о функциях `glibc`. Вместо этого для получения справки о функции вызывается соответствующая страница встроенной справочной системы Linux man.

В начале раздела были перечислены некоторые причины, заставляющие программистов прибегать к консольному программированию.

Здесь уместно указать еще одну. В ОС Linux принципы управления процессами, в частности механизмы создания и удаления дочерних процессов, отличаются от соответствующих механизмов ОС Windows. Для того чтобы создать дочерний процесс, будь то копия основного процесса или новый