Борис Пахомов

bhv

C/C++ и MS Visual C++ 2012

Основные элементы языков C/C++
Визуальная среда программирования
Создание основных типов приложений
Работа с наборами данных

ДЛЯ НАЧИНАЮЩИХ

Борис Пахомов

C/C++ и MS Visual C++ 2012 для начинающих

Санкт-Петербург
«БХВ-Петербург»
2013

УДК 004.4
ББК 32.973.26-018.2
П12

Пахомов Б. И.

П12 С/С++ и MS Visual C++ 2012 для начинающих. — СПб.: БХВ-Петербург, 2013. — 512 с.: ил.

ISBN 978-5-9775-0881-0

Книга является руководством для начинающих по разработке приложений в среде Microsoft Visual C++ 2012. Рассмотрены основные элементы языков программирования С/С++ и примеры создания простейших классов и программ. Изложены принципы визуального проектирования и событийного программирования. На конкретных примерах показаны основные возможности визуальной среды разработки Microsoft Visual C++, назначение базовых компонентов и процесс разработки различных типов консольных и Windows-приложений.

Для начинающих программистов

УДК 004.4
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 30.11.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 41,28.
Тираж 1800 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0881-0

Оглавление

Введение

Книга предназначена для начинающих программистов, поэтому многие вопросы раскрыты не во всей глубине, ибо читатель должен научиться пользоваться продуктом, а не тонуть в подробностях. Того, что дано здесь, достаточно для написания консольных приложений, а также многих приложений типа Windows Form. Автор надеется, что полученные знания явятся ступенью для изучения интересного и полезного (несмотря на его многие недостатки) программного продукта, каким, несомненно, является MS Visual C++ 2012. Для начинающего вполне достаточно изучить язык и Windows-формы, чтобы в дальнейшем, имея определенный багаж знаний, заниматься даже проблемами Интернета. Кстати, и в этом урезанном варианте некоторые выходы в Интернет имеются (например, через компонент `LinkLabel`).

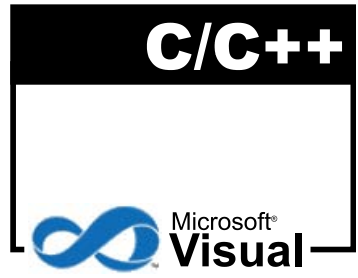
Примечание

В этой книге рассматривается программирование в Microsoft Visual C++ 2012, которая на этапе разработки и тестирования носила кодовое название Visual C++ 11.0 или Visual C++ 2011. Поскольку окончательная версия продукта появилась только в середине 2012 года, то она и получила название Microsoft Visual C++ 2012. При написании книги была использована пробная версия Microsoft Visual C++ 2011 Ultimate (в дальнейшем VC++).

Существенным недостатком этой версии, как и версий, начиная с 2008 г., является отсутствие возможности работы с базами данных в том варианте, как это было в версии 2005 г. Представитель разработчика еще в марте 2010 г. заверил, что указанный недостаток будет устранен. Но, к сожалению, и в этой новой версии 2011 г. все в плане работы с базами данных осталось по-старому. Более того, в данной версии отключена возможность работы с подсказчиком (надеюсь, что только потому, что она пробная), что вообще не поддается никакому объяснению.

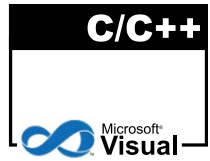
Однако все же есть небольшая лазейка в этом безнадежном деле: в *главе 12* приводится метод работы с базой данных типа MS Access на основе построения таблиц данных в самом приложении. Но и здесь стоит отметить очень плохого качества такие компоненты, как `dataSet`, `bindingSource` и особенно `dataGridView`: на одном наборе данных они могут работать, а на другом с такими же настройками компонентов, как и на предыдущем наборе, не работают.

Хочу отметить, что данная книга уточняет многие моменты, опущенные в прежнем издании, и дополнена новыми данными, например, по классам. Поэтому на ее фундаменте начинающему программисту будет значительно легче войти в среду VC++, нежели с помощью предыдущей книги автора.



ЧАСТЬ I

Изучение языка C/C++



ГЛАВА 1

Общие сведения о среде Visual C++ 2011. Создание консольного приложения

Общие положения

Вам требуется решить некоторую задачу. С помощью компьютера, конечно. Например, рассчитать движение материальных ценностей по некоторому складу: сколько чего было на данную дату, сколько чего поступило, сколько ушло, сколько осталось. С чего обычно начинают? Ясно сразу, что если задачу надо решать на компьютере, то следовало бы ее решение как-то формализовать, т. е. алгоритм ее решения (набор последовательных действий, исполнение которых приводит к решению задачи) требуется привести к последовательности неких формальных действий, понятных потом машине (говорят, что надо построить *машинный алгоритм решения задачи*). Затем надо продумать форму общения *(интерфейс)* того, кто станет решать эту задачу на компьютере *(пользователя)* с самим компьютером, исходя из максимального удобства общения. Это чуть ли не одна из главных трудностей проектирования решения задачи, ибо неудобство общения раздражает пользователя, который начинает совершать ошибки, что может привести, в конечном счете, к тому, что ваш проект просто будет отвергнут. Имеются еще некоторые шаги по подготовке к решению, но мы их здесь опустим, т. к. это не наша проблема.

Итак, мы изучили задачу, создали машинный алгоритм ее решения на компьютере, разработали интерфейс взаимодействия будущего пользователя с компьютером по решению данной задачи. А что дальше? А дальше все эти разработки надо перевести на понятный компьютеру язык, т. е., как говорят, *запрограммировать* наши действия, составить машинную программу, которая представляет собой последовательность определенных команд, записанных на выбранном для решения задачи языке (*алгоритмическом языке*, как принято называть). Для решения конкретной задачи с учетом разработанного интерфейса подходит не всякий алгоритмический язык. Поэтому разработчику и надо выбирать подходящий к данной ситуации язык, который бы отвечал требованиям к решению задачи. И не просто обеспечивал бы ее программирование, но и отвечал бы еще множеству других условий: позволял бы программисту быстро и надежно создавать программу, обеспечивал бы удобное сопровождение программы в период ее эксплуатации и т. д.

Когда программа написана на некотором алгоритмическом языке, она должна быть переведена в *машинный язык*, в язык, на котором работает компьютер, в его систему команд. Для этого существуют специальные программы, называемые *компиляторами*. Эти программы имеют параметры, задание которых позволяет компилятору создавать машинные программы в той или иной плоскости. Например, существуют параметры, позволяющие компилятору минимизировать размер памяти, которую станет занимать скомпилированная программа. Или, как мы будем рассматривать в данной книге, существует параметр, позволяющий компилятору создавать программы (часто говорят просто "коды"), состоящие из так называемых управляемых или неуправляемых кодов. Параметры компилятора называют по-разному: ключами, опциями.

Но программу мало откомпилировать. Компилирование — это только первый этап создания машинной программы. Дело в том, что в общем случае, для решения конкретной задачи, т. е. реализации ее машинного алгоритма, требуется подключение неких стандартных *библиотек*, содержащих стандартные программы, которые разрабатываются один раз и используются во многих алгоритмах. Например, перевод десятичных чисел в двоичные или шестнадцатеричные. Каждый программист, пишущий программу, не станет всякий раз заниматься этим переводом. Поэтому в подобных случаях разработчик программной среды, в рамках которой создается программный продукт (в нашем случае — это Visual Studio 2011), сам создает подобные библиотеки и поставляет их со средой разработки, которая содержит и компиляторы с разных языков среды в машинные коды. В свою очередь, и опытный программист может самостоятельно создавать такие библиотеки и включать их в общий перечень библиотек среды программирования, чтобы они в дальнейшем подключались автоматически к решению задач. Для этой цели среда поставляет специальные средства.

Но вернемся к компиляции. Компилятор, просматривая программу (код, как говорят), переводит ее (код) в машинные команды. Но не просто в набор команд, а формирует это все множество в виде отдельных (*объектных*) *модулей*. В каждом таком модуле создается своя таблица имен со ссылками на их месторасположение. То есть компилятор создает не исполняемый код, а так называемый *объектный код*, содержащий неконкретные (как говорят "неразрешенные", т. е. неопределенные) ссылки. На этом его работа завершается. Чтобы получить *исполняемый модуль*, надо "разрешить", т. е. конкретизировать ссылки, сформированные компилятором с учетом конкретного размещения данного кода в выделенной для него памяти компьютера. Эту работу выполняет специальная программа, которая называется по-разному: *редактор связей*, *компоновщик*, *линковщик*, *построитель*. После работы этой программы получается набор машинных команд, готовых к исполнению на компьютере.

В свете всего вышесказанного можно утверждать, что для того чтобы создать программу, требуется еще иметь средства ее компиляции и компоновки. Эти средства поставляются в рамках изучаемой нами среды MS Visual C++ 2011. То есть, прежде чем перейти к изучению собственно языка C/C++, надо познакомиться хотя бы в общих чертах с интерфейсом среды обработки данных MS Visual C++ 2011, что-

бы иметь возможность с его помощью компилировать и строить исполняемые программы в рамках языка C/C++.

Структура рабочего стола среды программирования

Цель этой главы — продемонстрировать начальные элементы программирования на языке C/C++. Язык C++ является дальнейшим развитием языка C, поэтому все конструкции языка C поддерживаются в C++. Однако в C++ появились новые возможности синтаксиса, не имеющиеся в C. Это мы увидим по мере рассмотрения материала.

Чтобы построить программу на этом языке, нам надо воспользоваться средой программирования Visual C++ 2011 (ее английская аббревиатура — IDE: Integrated Development Environment), которая содержит средства создания программы, ее компиляции, отладки и запуска на выполнение. В этой связи рассмотрим кратко структуру этой среды, а точнее, ее интерфейс с нами, пользователями. Интерфейс — это аппарат, который позволяет удобно взаимодействовать пользователю со средой.

После установки на своем компьютере среды Visual C++ 2011 она запускается на выполнение. Здесь следует отметить один момент.

Среда Visual Studio 2011 Professional состоит из многих подсред, подразделов, каждый из которых нацелен функционально обрабатывать определенную область задач. Чтобы настроиться на работу с конкретным подразделом, разработчики предусмотрели для пользователя возможность выбора одной конкретной подсреды, с которой он станет работать после запуска основной среды. Когда пользователь выберет такую подсреду (в данном случае автор выбрал подсреду Visual C++), основная среда настраивается на выбранный подраздел и пользователь получает к нему доступ. Адрес загрузки такой подсреды, как и при установке любой программы, попадает в меню кнопки **Пуск**, откуда вы можете ее загрузить, воспользовавшись командой главного меню **Пуск | Программы**. Для удобства дальнейшей работы с установленным программным продуктом следует мышью перетянуть его значок  на линейку быстрого запуска программ, которая находится на рабочем столе операционной системы (обычно ее располагают в нижней части стола). Находящийся на этой линейке любой программный продукт запускается одинарным щелчком мыши на значке соответствующего продукта. Итак, загружаем наш продукт Microsoft Visual C++ 2011 (для краткости в дальнейшем станем его называть VC++). На экране появится главное окно — рабочий стол, структуру которого мы и рассмотрим.

Главное окно

Общий вид окна показан на рис. 1.1. Этот формат интерфейса принят в среде по умолчанию и может быть всегда восстановлен через соответствующую опцию главного меню: **Windows | Reset Window Layout.**

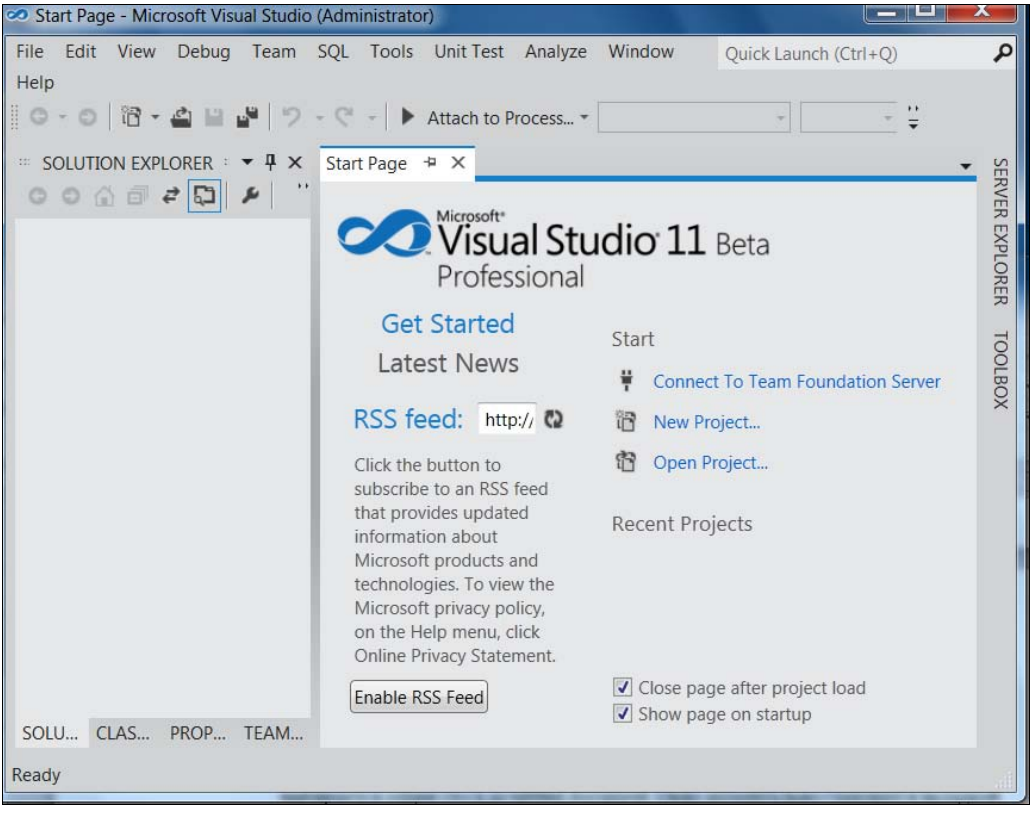


Рис. 1.1. Вид главного окна IDE после загрузки VC++

В верхней части окна расположена строка с *командами главного меню* среды (команды: **File**, **Edit**, ...) — это строка горизонтального меню. При вызове этих команд (их еще называют опциями, т. е. элементами выбора из нескольких значений) открываются так называемые *"выпадающие меню"* — это вертикальные меню, представляющие собой набор команд, располагающихся на экране сверху вниз. Пример такого меню показан на рис. 1.2.

Ниже строки главного окна находятся *кнопки быстрого вызова* некоторых команд на исполнение. Все эти кнопки имеют всплывающие подсказки (надо навести курсор мыши на кнопку, немного подождать, после чего появится подсказка о том, для чего предназначена данная кнопка). Рядом с такими кнопками могут быть дополнительные кнопки для раскрытия списка значений основной кнопки. Так как все кнопки не помещаются в отведенное им место на рабочем столе, то они свернуты в небольшие полосы с кнопками их развертывания точно так же, как это выполнено во всем известной программе Word (рис. 1.3).

Вид главного окна, в свою очередь, изменяется при задании типа создаваемого приложения. С этим мы познакомимся, когда начнем создавать приложения.

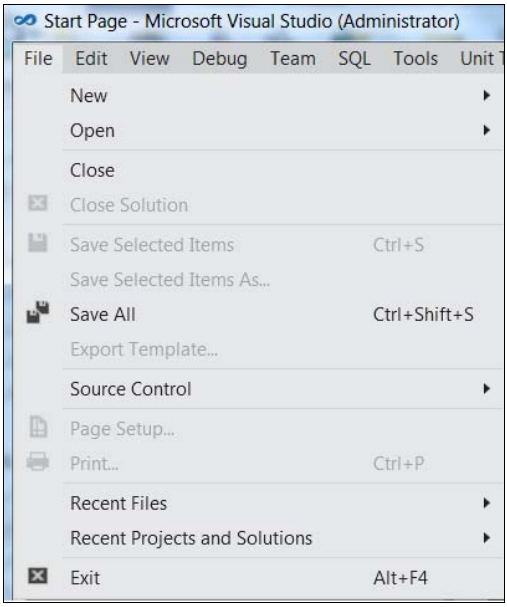


Рис. 1.2. Пример выпадающего меню

Рис. 1.3. Кнопки быстрого вызова

Некоторые замечания

О рабочем столе

Рабочий стол формируется из набора окон. Каждое окно — это обычное Windows-окно, имеющее стандартную заголовочную полосу в своей верхней части. За эту полосу можно окно перемещать с помощью протягивания мышью. У окон имеются свойства, которые открываются, если на заголовочной части щелкнуть правой кнопкой мыши. Перечень свойств окна **Solution Explorer** показан на рис. 1.4.

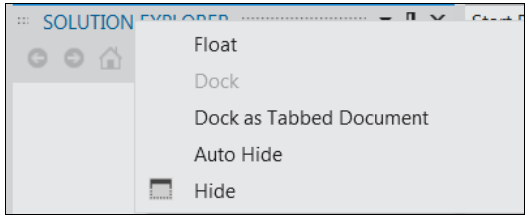


Рис. 1.4. Меню свойств окна

Кстати сказать, стартовая страница, исключая главное меню и набор кнопок быстрого запуска, — это тоже обычное окно со своим набором свойств.

Особенностью набора окон является возможность разбирать этот набор на подокна и группировать последние между собой в соответствии с желаниями пользователя. Для этого у каждого окна имеются определенные свойства. Чтобы лучше понять,

как работают некоторые свойства окон, введем такое понятие, как *причаливание*. По самому смыслу этого слова понятно, что объект, обладающий свойством причаливания, может по аналогии с морским или речным портом причаливать либо к берегу, либо к другому объекту. В нашем случае таким берегом является полоса стартовой страницы, содержащая главное меню и кнопки быстрого запуска. К ней могут причаливать другие окна, образуя у этого "берега" свои вкладки, как это делается в интернет-браузерах, когда открывается та или иная Web-страница. Но причаливание окна может осуществляться и к другому окну или к группе окон, образующих из своих вкладок новый берег, когда одно или несколько окон занимают всю горизонтальную полосу страницы, а остальные окна располагаются в нижней полосе, образуя своими вкладками свой "берег причаливания". При этом окно, к которому причаливает другое окно, как бы захватывает, проглатывает причаливающее к нему окно, оставляя от последнего только вкладку с именем "проглоченного" окна, по которой можно открыть такое причалившее и "проглоченное" окно.

Как визуально определить, захватится ли перемещаемое вами окно "берегом" или другим окном-объектом? Когда вы перемещаете по экрану окно, вы видите некие направляющие "кресты", возникающие на вашем пути и синие подсветки, всплывающие в некоторых областях перемещения. Кресты возникают по центру активной вкладки, через территорию которой вы перемещаетесь. *Вкладка* — это какое-то окно, причаленное к берегу или объекту. Синяя подсветка же появляется в момент, когда подсвеченная область готова захватить ваше перемещаемое окно, как только вы отпустите левую кнопку мыши. Подсветки возникают справа, слева, сверху, снизу у креста, показывая область размещения будущего захваченного объекта. Уточним, чтобы "вытащить" окно из какой-то группы, надо зацепиться мышью за его заголовок и начать протяжку окна. Если получилось так, что окна расположились горизонтально в несколько слоев (например, первый слой содержит вкладки двух окон, второй — двух окон, третий — одного окна), а вы хотите, чтобы все вкладки были в одном слое, вытащите, ухватив мышью заголовок вкладки, окно в область, где оно не попадает на синее пространство, уменьшите мышью горизонтальный размер окна и снова вставьте окно (через его попадание в поле, в котором появится синяя подсветка) в место его причаливания. Так поступите с каждым из окон во всех слоях, добиваясь, чтобы они по ширине заголовка помещались в причаливамый "берег". Если вы случайно закрыли какое-то окно (оно при этом, естественно, исчезло с экрана), следует найти его с помощью опций главного меню **Windows** и **View**, на которых надо просто щелкнуть мышью.

Вот опции окна (открываются правой кнопкой мыши):

- **Float** (плавающее). Такое окно можно перетягивать в любую часть рабочего стола. Оно имеет вид обычного Windows-окна. Подобное окно как бы отвязано от набора окон и может перемещаться мышью в любое место экрана. При этом, как только протяжка окна завершается (напомним, что протяжка — это захват окна за его заголовочную полосу мышью и удержание левой кнопки мыши в момент продвижения окна по экрану; завершение протяжки — отпускание нажатой левой кнопки мыши), окно остается в том же месте, где оно находилось в момент завершения протяжки. Но так получается только тогда, когда окно в мо-

мент остановки не попало на поле синей подсветки, иначе окно захватится и причалит к области, определенной синей подсветкой;

- **Dock** (причаливающее). Вспомните морское понятие "док" — место для причаливания судов. Это аналогия с нашим случаем. Окно, у которого выбрана подобная опция, моментально причалит к какому-то "берегу" (зависит от его местоположения в момент выбора опции **Dock**);
- **Dock as Tabbed Document** (причаливать в качестве вкладки к основному причалу рабочего стола — к группе опций главного меню). Как только мы выбираем эту опцию у какого-то окна, оно моментально причаливает к основному причалу и становится вкладкой, располагающейся первой слева. Таким способом можно переупорядочить последовательность вкладок, назначая каждой в заданном порядке рассматриваемое свойство окна.

Пример показан на рис. 1.5. На рис. 1.5, *а* приведен вид рабочего стола VC++, в котором четыре окна представлены в виде своих вкладок, расположенных под полем главного меню (в данном случае оно будет играть роль причала). С помощью опции **View** главного меню (**View | Other Windows | Server Explorer**) выбрано окно **Server Explorer**, которое появилось на экране и свойства которого показаны на рис. 1.5, *б*. Затем выбирается опция **Dock as Tabbed Document**. Окно моментально становится вкладкой того причала, который показан ранее на рис. 1.5, *а*. В результате имеем вид уже пяти окон, приведенный на рис. 1.5, *в*;

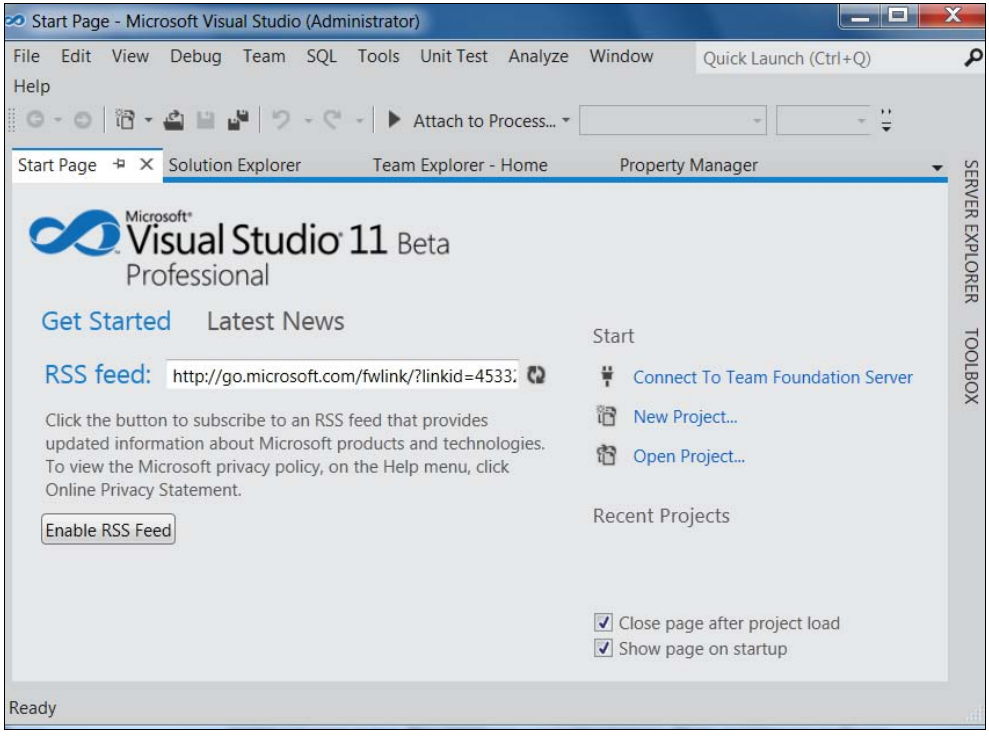
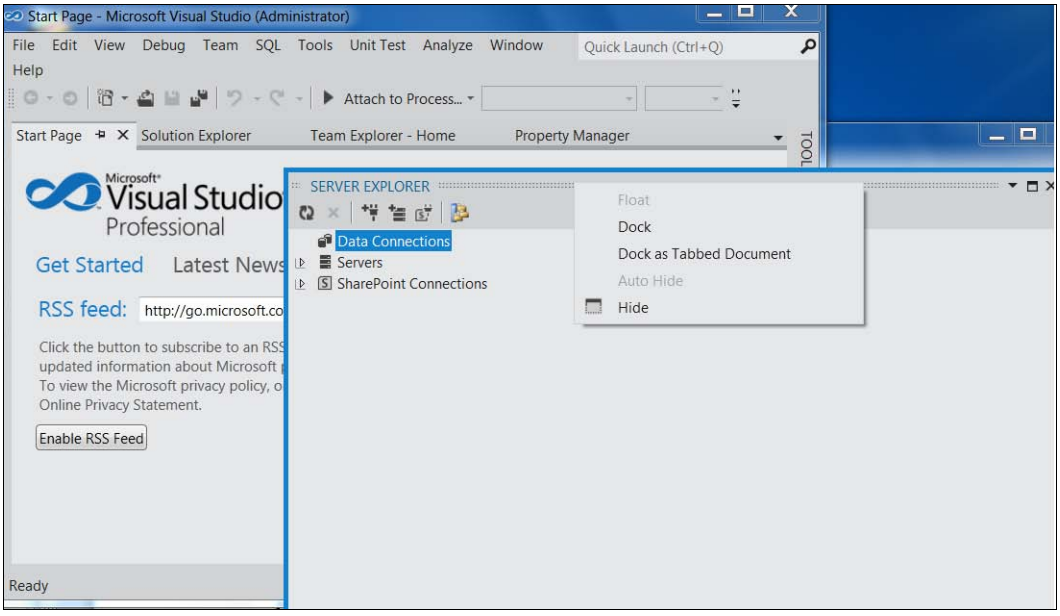
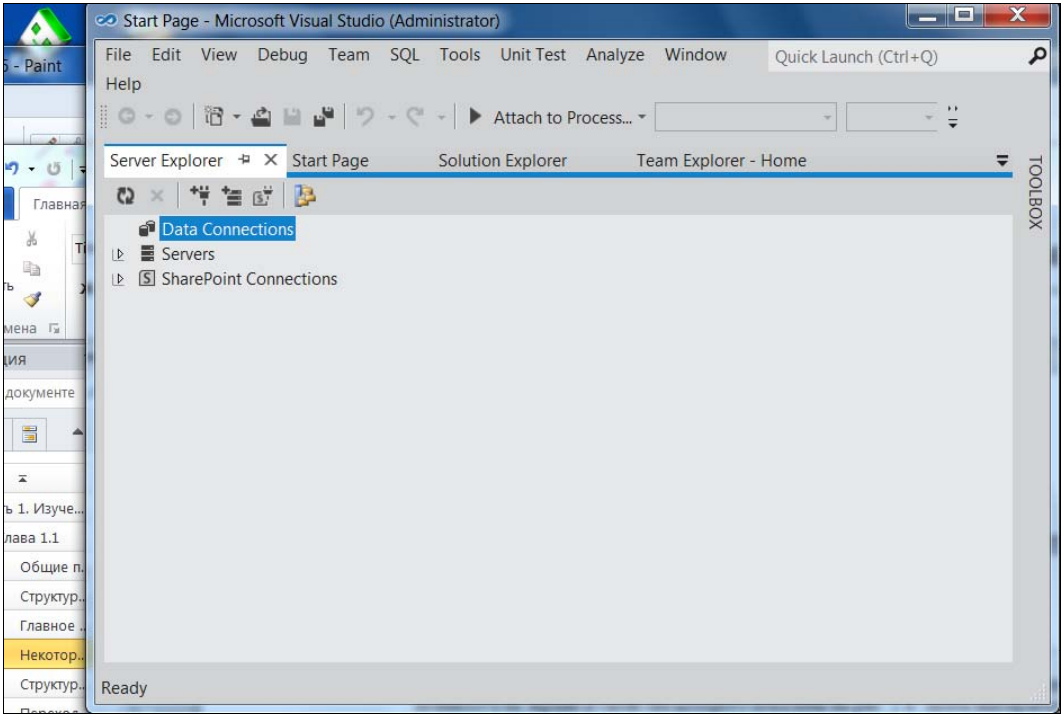


Рис. 1.5. Изменение рабочего стола VC++: *а* — первоначальный вид VC++



б



в

Рис. 1.5. Изменение рабочего стола VC++:
б — окно **Server Explorer** и его свойства-опции;
в — окно **Server Explorer** после выбора его свойства **Dock as Tabbed Document**

- **Auto Hide** (автоматически исчезать). В этом случае окно автоматически "прячется" (причаливает) в качестве вкладки к ближайшей боковой стороне основного окна рабочего стола;
- **Hide** (спрятать). При выборе этого свойства окно исчезает с экрана. Чтобы оно снова появилось, надо воспользоваться либо опцией **View** главного меню, либо соответствующей данному окну опцией **Other Windows** опции **View.**

Все рассмотренные выше свойства надо хорошо понимать, чтобы манипулировать положением окон на рабочем столе, иначе может сложиться ситуация, когда на рабочем столе соберется множество окон, которые просто станут мешать работать. Их надо будет "разогнать" по боковым сторонам основного окна, а другие просто спрятать. Например, как установить справа сбоку основного окна рабочего стола окна **Toolbox** и **Server Explorer**? Предварительно надо щелкнуть мышью на вкладке окна, и оно появится на рабочем столе. Тогда у него можно увидеть его опции. У рассматриваемых окон надо выбрать (через правую кнопку мыши) свойство **Dock** (причаливание). Тогда в заглавной строке окна появится значок **Auto Hide** (скрывать автоматически). Значок имеет вид вертикального полупроводника. Если на этом значке щелкнуть, то окно причалит к правой стороне главного окна среды и спрячется: останется видно только его имя.

О справочной системе Help

Все всегда начинается с вопроса: "А где посмотреть, как это делается?" Ясно, что в справочной системе к программному продукту. Откроем опцию **Help** главного меню VC++ (рис. 1.6).

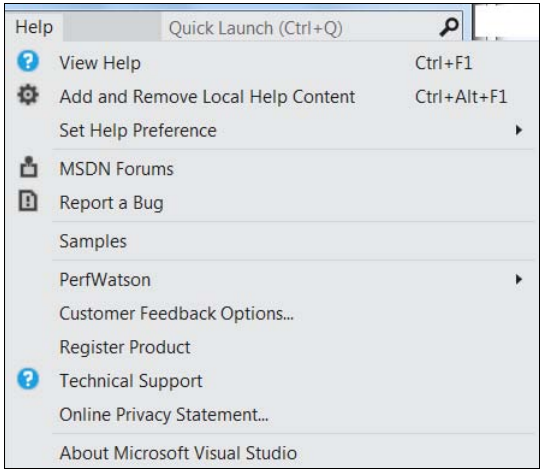


Рис. 1.6. Меню **Help**

В рассматриваемой нами среде программирования VC++ предусмотрено использование справочной информации из двух источников: из Интернета и из своего компьютера. Для этого справочную систему следует предварительно настроить на желаемый источник информации.

Почему из двух источников? Если иметь справку на своем компьютере, ее можно очень быстро загружать. Но за это надо платить: приходится самому поддерживать

ее в актуальном состоянии. Для этого в меню **Help** служит опция **Add and Remove Local Help Content**. Если же пользоваться базой данных разработчика среды, то база данных, естественно, поддерживается самим разработчиком, что удобно. Но и за это удобство есть своя плата: приходится загружать справку через Интернет, что, во-первых, дольше и дороже, а во-вторых, Интернет может быть в нужный момент и недоступен.

Настройка источника информации осуществляется через опцию **Set Help Preference** меню **Help** (рис. 1.7).

Вот теперь, когда вы начнете выполнять опцию **View Help**, показанную на рис. 1.6, ваш браузер откроет вам доступ к справочной системе — рис. 1.8.

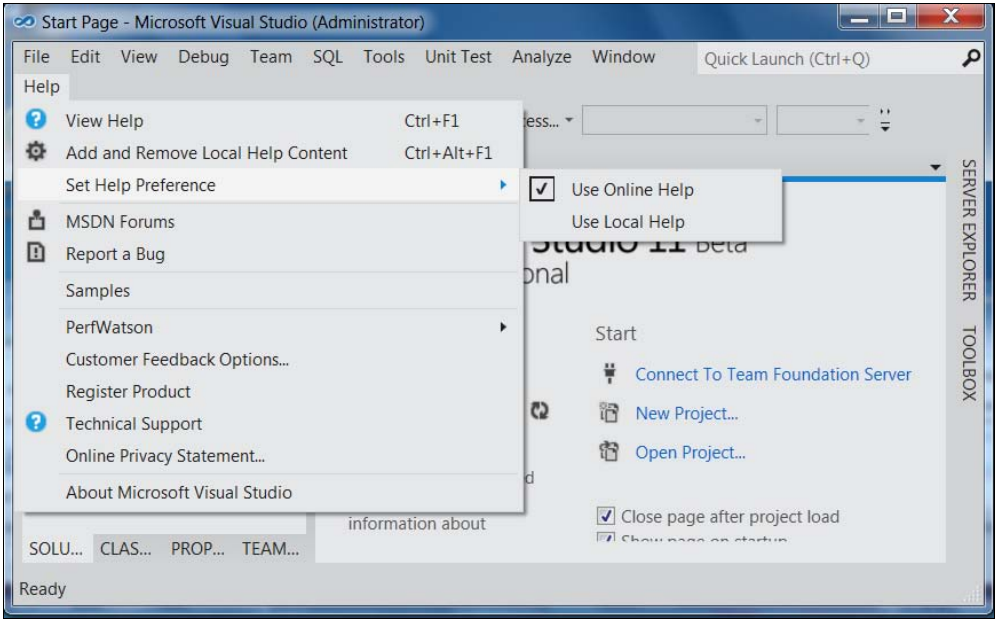


Рис. 1.7. Настройка среды программирования на источник справочной информации

Рис. 1.8. Окно MSDN

Обратим внимание на опцию **Samples** (образцы) в подменю опции Help, которая открывает доступ к примерам приложений различного направления, заготовленным разработчиком.

Структура программ в VC++

Программы в VC++ называются приложениями (очевидно, приложениями к среде IDE). Мы так и дальше станем их называть. Приложения строятся средой в виде специальных конструкций — проектов, которые выглядят для пользователя как совокупность нескольких файлов.

Программа на языке C, расширением которого является язык C++ (далее не станем их пока разделять и будем писать C/C++), — это совокупность функций, т. е. специальных программных образований, отвечающих определенным требованиям. Причем приложение — это главная функция, внутри которой помещаются операторы, реализующие алгоритм приложения. Среди операторов имеются такие, которые служат для вызова других функций, требующихся при реализации алгоритма. Запуск любой программы начинается с запуска *главной функции*, содержащей всю остальную часть программы. Часть функций создается самим программистом, другая часть — библиотечные функции — поставляется пользователю со средой программирования и используется в процессе разработки программ. При изучении C/C++ мы будем пользоваться специальным видом приложений — консольными приложениями, которые формируются на основе заранее заготовленных в среде проектирования шаблонов.

Консольные (т. е. опорные, базовые) *приложения* — это приложения без графического интерфейса, которые взаимодействуют с пользователем через специальную командную строку или (если они работают в рамках IDE) запускаются специальной командой из главного меню среды. Такие приложения создаются с помощью специального шаблона, доступного из диалогового окна, открывающегося после выполнения команды **File | New | Project.**

Шаблон консольного приложения добавляет в создаваемое приложение необходимые элементы (создается заготовка будущего приложения), после чего разработчик вставляет в этот шаблон свои операторы на языке C/C++. Затем приложение компилируется в автономный исполняемый файл и может быть запущено на выполнение. Общение с пользователем происходит через специальное так называемое консольное окно, открывающееся средой после запуска приложения (в это окно выводятся сообщения программы, через него вводятся данные для расчета и в него же выводятся результаты расчетов).

Компиляция и сборка проекта осуществляется через опцию **Build** главного меню среды (структура главного меню меняется в зависимости от совершаемых действий: при загрузке среды она одна (в ней нет опции **Build**), а например, если мы создадим проект, такая опция появляется). После компиляции и сборки проект можно запустить на выполнение. Запуск на выполнение осуществляется с помощью опции **Debug** главного меню среды.

Изучение C/C++ мы станем осуществлять на примерах: будем создавать программы, разбирать, как они работают с параллельным изучением их структуры. Консольные приложения, которые мы начнем создавать, имеют шаблон вида CLR Console Application. Последние два слова этого названия понятны — консольное приложение. А что означает аббревиатура CLR? В настоящей редакции среды разработки авторы создали два варианта консольного приложения, только разнесли их использование по разным диалоговым окнам: шаблон с CLR задан в папке CLR, а шаблон для обычного консольного приложения задан в папке Win32.

Так что же такое — CLR (Common Language RunTime)? Это специальная среда, которая управляет исполнением программного кода, памятью, потоками данных и работой с удаленными компьютерами, при этом строго обеспечивая безопасность и создавая надежность исполнения кода. CLR является добавкой-расширением C++, введенной фирмой Microsoft, начиная с версии VC++ 2005. В версиях 2005, 2008 подключение к вашему проекту этой среды осуществлялось на этапе компиляции. Там для консольного приложения (по умолчанию) этот режим не поддерживался (т. е. консольное приложение как бы оставалось в старой версии C++, в "родной" (native), как определили ее авторы добавки CLR). В этой старой версии, когда вы в своей программе работали с некоторыми объектами (здесь нам приходится забегать вперед, ибо объекты — это предмет более позднего изучения, когда мы станем знакомиться с классами), то должны были сами заботиться об их размещении в памяти, выделяемой средой. Память для вашего приложения выделяется в так называемой *куче*: в ней вы размещаете свои объекты, там же сами освобождаете память, когда перестаете работать с объектом, иначе куча может переполниться и процесс выполнения приложения прервется. Это так называемая неуправляемая куча. Указатели (о них — позже) на участки памяти в такой куче обозначаются символом "*".

Другое дело, когда включается режим CLR. Такое приложение отличается от обычного тем, что его заготовка обеспечивает подключение к приложению специального системного пространства `System`, содержащего объекты, размещение в памяти которых надо автоматически регулировать. Так вот: режим CLR работает уже с управляемой кучей памяти, в которой размещение объектов и ее освобождение от них происходит под управлением среды. Такой сервис входит в язык Java, где не надо делить кучу на управляемую и неуправляемую. В этой среде употребляются так называемые регулируемые указатели на объекты.

Регулируемый указатель — это тип указателя, который ссылается на объекты (адреса памяти, по которым можно обращаться к объектам), расположенные в общей регулируемой куче памяти, предоставленной приложению в момент его исполнения. Для таких указателей принято специальное обозначение: вместо символа "*" применяется символ "^".

Создание CLR привело к необходимости разработки аппарата преобразования переменных, относящихся к одной куче, в адреса в другой куче и т. п. Этот процесс назвали *маршализацией*. Существует специальная библиотека, обеспечивающая этот процесс. Однако все это довольно усложнило программирование.

Переход к созданию консольного приложения

Для создания консольного приложения воспользуемся шаблоном CLR-приложения. Для этого необходимо выполнить следующие шаги:

1. Загрузить среду VC++.
2. Выполнить команды главного меню **File | New | Project**. Откроется диалоговое окно, показанное на рис. 1.9. В этом окне выберите опцию **CLR Console Applicaton**, задайте в его нижней части имя будущего проекта в поле **Name**, которое потом перекочует в поле **Solution name**. С помощью кнопки **Browse** установите папку, в которую будет помещен ваш проект. Теперь окно рис. 1.9 станет выглядеть так, как показано на рис. 1.10.

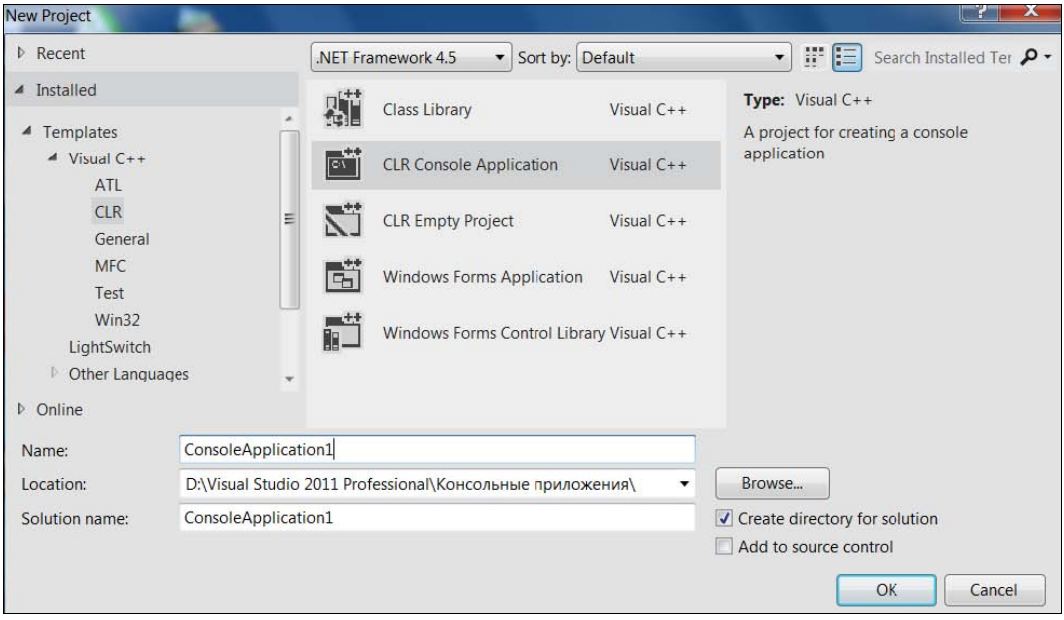


Рис. 1.9. Выход на шаблон создания консольного приложения

3. Нажать кнопку **OK**. В результате получится то, что показано на рис. 1.11.

Заготовка консольного приложения состоит из заголовка главной функции

```
int main(array<System::String ^> ^args)
```

и тела, ограниченного фигурными скобками.

Преобразуем заголовок функции `main(множество аргументов функции)` к виду `main()`, т. е. к виду без аргументов, а из тела удалим оператор `return 0`.

Все это проделаем с помощью Редактора кода, который открывается одновременно с появлением заготовки консольного приложения на экране (заготовка сразу помещается в поле Редактора кода). Чтобы убедиться, что вы находитесь в Редакторе, щелкните кнопкой мыши в любом месте поля заготовки и увидите, что курсор

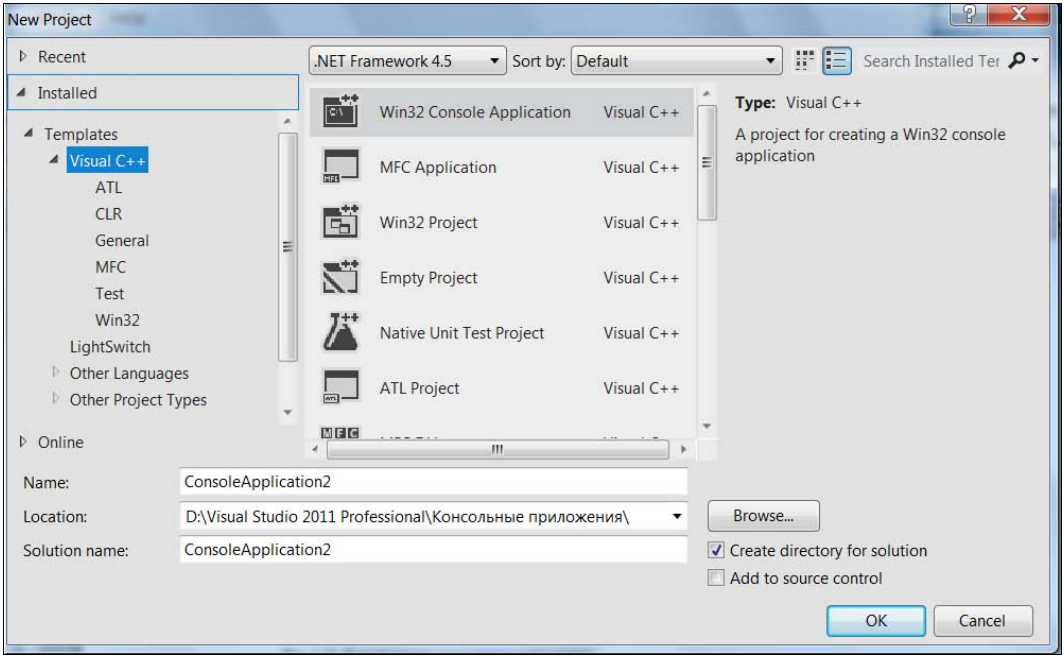


Рис. 1.10. Формирование консольного приложения

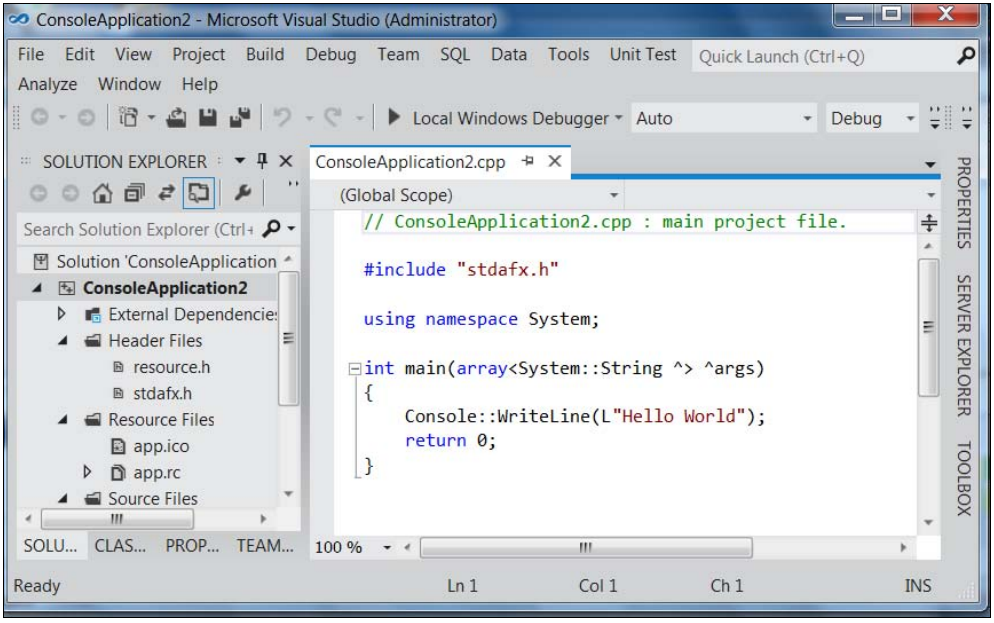


Рис. 1.11. Вид заготовки консольного приложения

установится в месте вашего щелчка (Редактор ждет в этой точке ваших дальнейших действий). Далее можно набирать любой текст, как в обычном современном текстовом редакторе, работать клавишами <Delete>, <Backspace>, клавишами-стрелками и другими необходимыми для ввода и редактирования клавишами.

Итак, мы привели заголовок функции `main(аргументы)` к виду `main()`. Это означает, что наша главная функция не будет иметь аргументов, которые служат для связки между собой нескольких консольных приложений. Этим мы заниматься не будем.

Когда мы формировали заготовку консольного приложения, мы видели, что при задании имени (**Name**) приложения формировалось и некое поле **Solution name** (решение). Дело в том, что среда VC++ оформляет создаваемое приложение в виде двух контейнеров, вложенных один в другой. Один (главный контейнер) называется **Solution** (решение), а другой — **Project** (проект). Проект определен как конфигурация (каркас, контейнер), объединяющий группу файлов.

В рамках проекта создается программа, в том числе и подлежащая исполнению, т. е. откомпилированная и построенная. Каждый проект содержит по крайней мере две подконфигурации: отладочную и обычную (исполнительскую). Это задается в выпадающем меню, которое по умолчанию установлено на опцию **Debug** (отладка). Это выпадающее меню находится в строке окна среды, расположенной ниже строки главного меню (рис. 1.12).

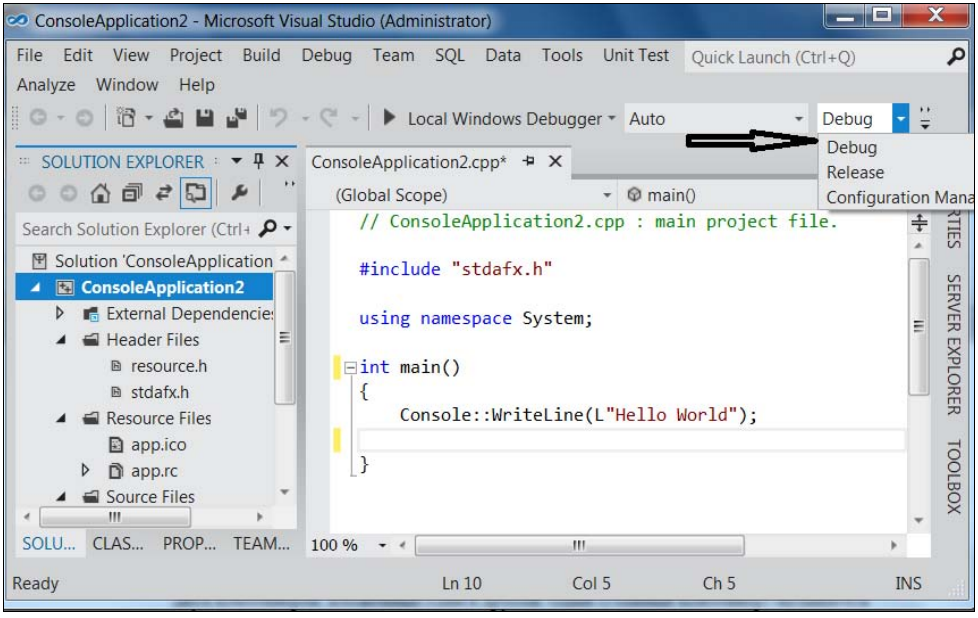


Рис. 1.12. Установка режима исполнения программы

Проекты являются частью другого каркаса, другого контейнера, который называется **Solution** (решение) и который отражает взаимосвязь между проектами: одно решение может содержать множество проектов, а проект содержит множество элементов, обеспечивающих существование приложения как такового. Можно сказать,