Лабораторная работа №2

«Знакомство со средой разработки Microsoft Visual Studio»

Цель работы: изучить нструментальную среду разработки приложений Microsoft Visual Studio 2010 в режиме компилятора языка С на примере создания простого консольного приложения.

Задание на выполнение см. тут.

Теоретическая часть

Язык **С** (читается как *Си*) в основе своей был создан в 1972 г. как язык для операционной системы *UNIX*. Автором этого языка считается Денис М. Ритчи (DENNIS M. RITCHIE).

Популярность языка **С** обусловлена, прежде всего тем, что большинство операционных систем были написаны на языке С. Его начальное распространение было задержано из-за того, что не было удачных компиляторов.

Несколько лет не было единой политики в стандартизации языка **С**. В начале 1980-х г. в Американском национальном институте стандартов (*ANSI*) началась работа *по* стандартизации языка **С**. В 1989 г. работа комитета *по* языку С была ратифицирована, и в 1990 г. был издан первый официальный документ *по* стандарту языка **С**. Появился стандарт 1989, т.е. **С89.**

К разработке стандарта *по* языку **С** была также привлечена Международная организация *по* стандартизации (*ISO*). Появился стандарт *ISO*/IEC 9899:1990, или *ANSI* C99 языка **С.**

В данном пособии за основу принимается *стандарт языка* **С** от 1989 г. и написание программ будет выполняться в среде разработки *Visual Studio* 2010.

Язык **С** является прежде всего языком высокого уровня, но в нем заложены возможности, которые позволяют программисту (пользователю) работать непосредственно с аппаратными средствами компьютера и общаться с ним на достаточно низком уровне. Многие *операции*, выполняемые на языке **С**, сродни языку Ассемблера. Поэтому язык С часто называют языком среднего уровня.

Для написания программ в практических разделах данного учебного пособия будет использоваться *компилятор* языка **С++**, а*программирование* будет вестись в среде **Microsoft Visual Studio 2010**. Предполагается, что на компьютере установлена эта интегрированная среда.

Microsoft Visual Studio 2010 доступна в следующих вариантах:

- **Express** – бесплатная среда разработки, включающая только базовый набор возможностей и библиотек.
- **Professional** – поставка, ориентированная на профессиональное создание программного обеспечения, и командную разработку, при которой созданием программы одновременно занимаются несколько человек.
- **Premium** – издание, включающее дополнительные инструменты для работы с исходным кодом программ и создания баз данных.

- **Ultimate** – наиболее полное издание Visual Studio, включающие все доступные инструменты для написания, тестирования, отладки и анализа программ, а также дополнительные инструменты для работы с базами данных и проектирования архитектуры ПО.

Отличительной особенностью среды **Microsoft Visual Studio 2010** является то, что она поддерживает работу с несколькими языками программирования и программными платформами. Поэтому, перед тем, как начать создание программы на языке С, необходимо выполнить несколько подготовительных шагов *по* созданию проекта и выбора и настройки компилятора языка С для трансляции исходного кода

После запуска **Microsoft Visual Studio 2010** появляется следующая стартовая страница, которая показана на рис. 1.1.

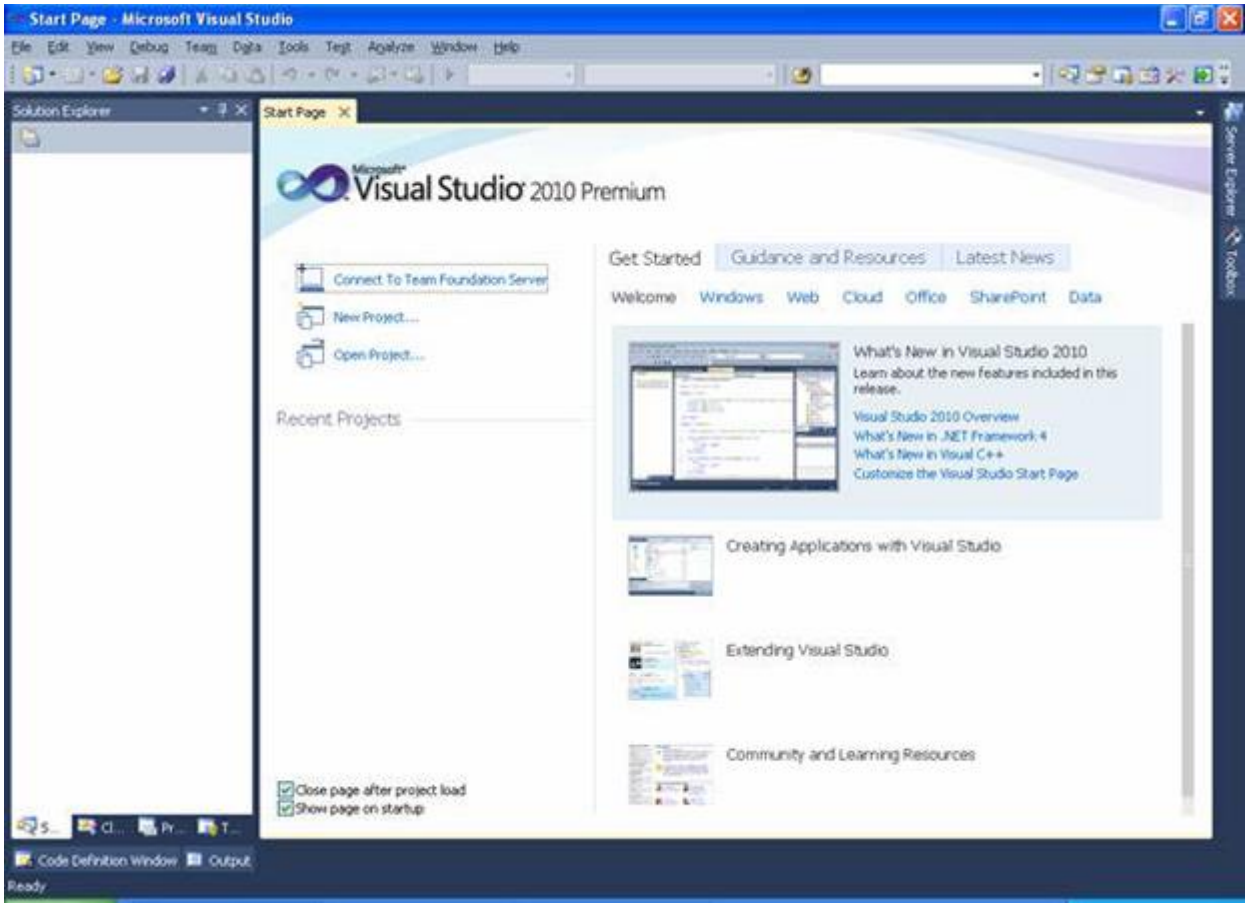

Рис. 1.1. Стартовая страница Visual Studio 2010

Следующим шагом является создание нового проекта. Для этого в *меню* **File** необходимо выбрать **New – Project** (или комбинацию клавиш **Ctrl + Shift + N**). Результат выбора пунктов *меню* для создания нового проекта показан на рис. 1.2.

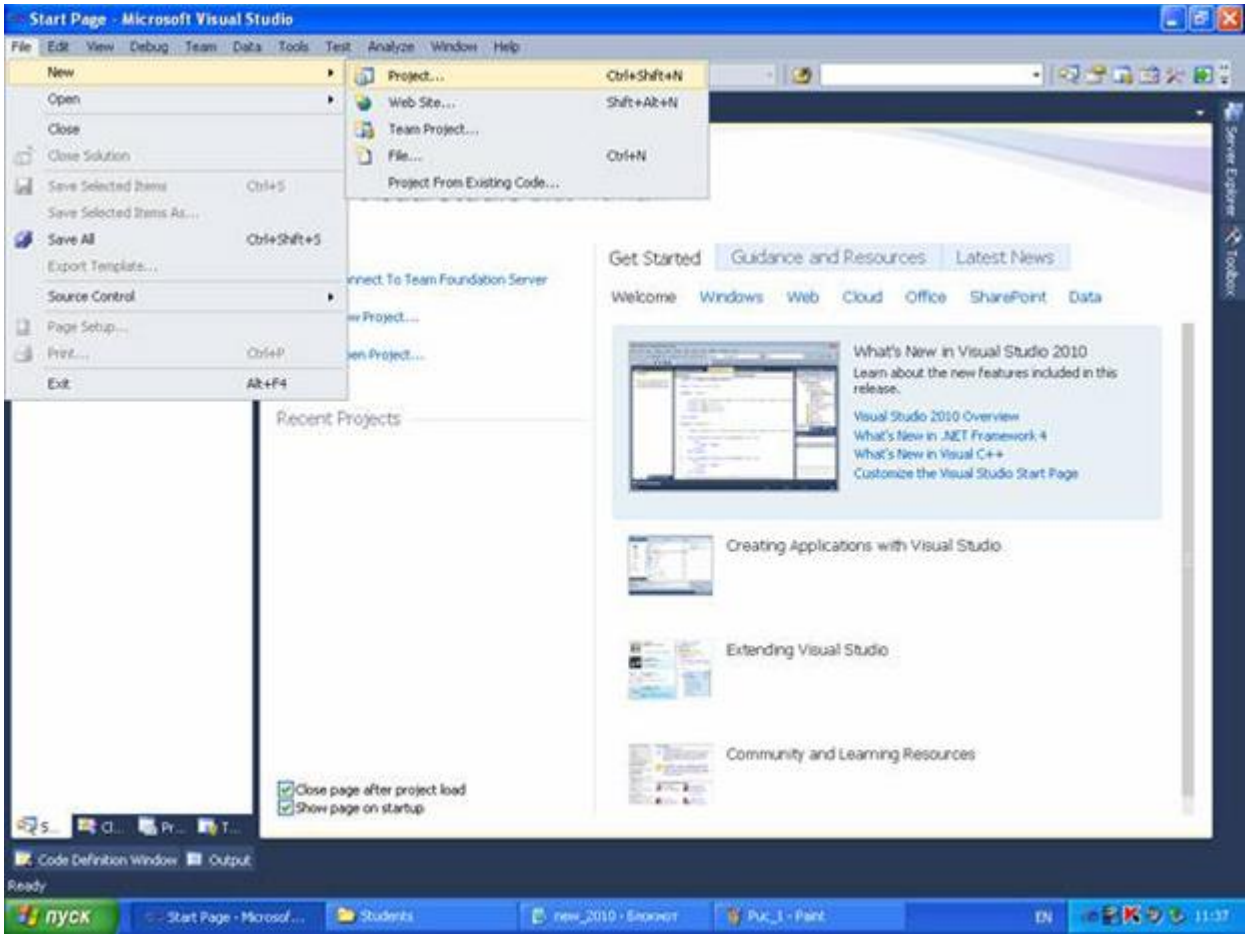

Рис. 1.2. Окно с выбором нового проекта

Среда *Visual Studio* отобразит окно **New Project**, в котором необходимо выбрать тип создаваемого проекта. *Проект* (*project*) используется в *Visual Studio* для логической группировки нескольких файлов, содержащих исходный код, на одном из поддерживаемых языков программирования, а также любых вспомогательных файлов. Обычно после сборки проекта (которая включает компиляцию всех входящих в проект файлов исходного кода) создается один исполняемый *модуль*.

В окне **New Project** следует развернуть узел *Visual* C++, обратиться к пункту Win32 и на центральной панели выбрать Win32 *Console Application*. Выбор этой опции показан на рис. рис. 1.3.

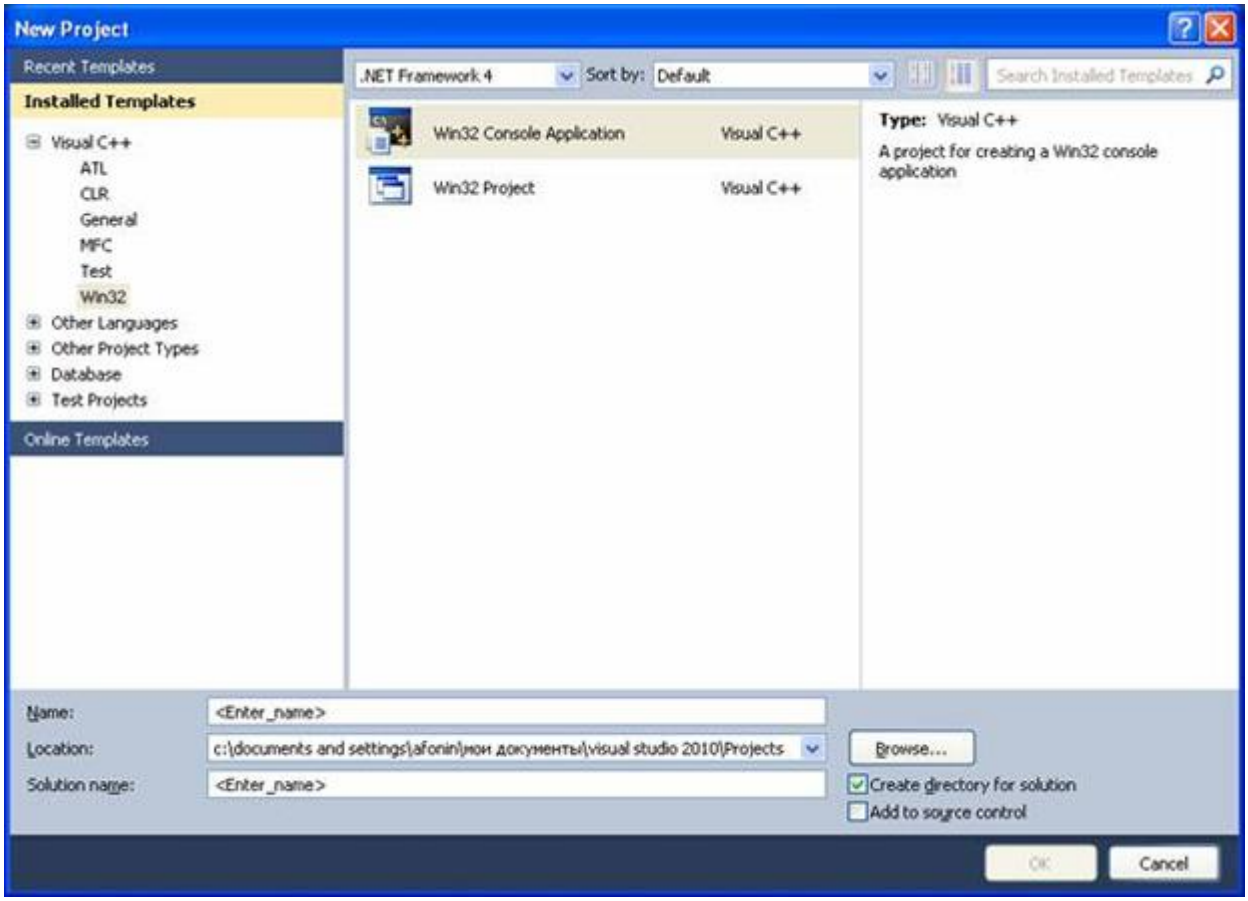

Рис. 1.3. Выбор типа проекта

Затем в *поле* редактора **Name** (где *по* умолчанию имеется <Enter_name>) следует ввести имя проекта, например, **hello**. В *поле***Location** можно указать *путь* размещения проекта, или выбрать *путь* размещения проекта с помощью клавиши (кнопки) Browse. *По*умолчанию проект сохраняется в специальной папке Projects. Пример выбора имени проекта показано на рис. 1.4.

Одновременно с созданием проекта *Visual Studio* создает решение. *Решение* (solution) – это способ объединения нескольких проектов для организации более удобной работы с ними.

После нажатия кнопки **OK** откроется окно **Win32 Application Wizard** (мастер создания приложений для операционных систем *Windows*), показанное на рис. 1.5.

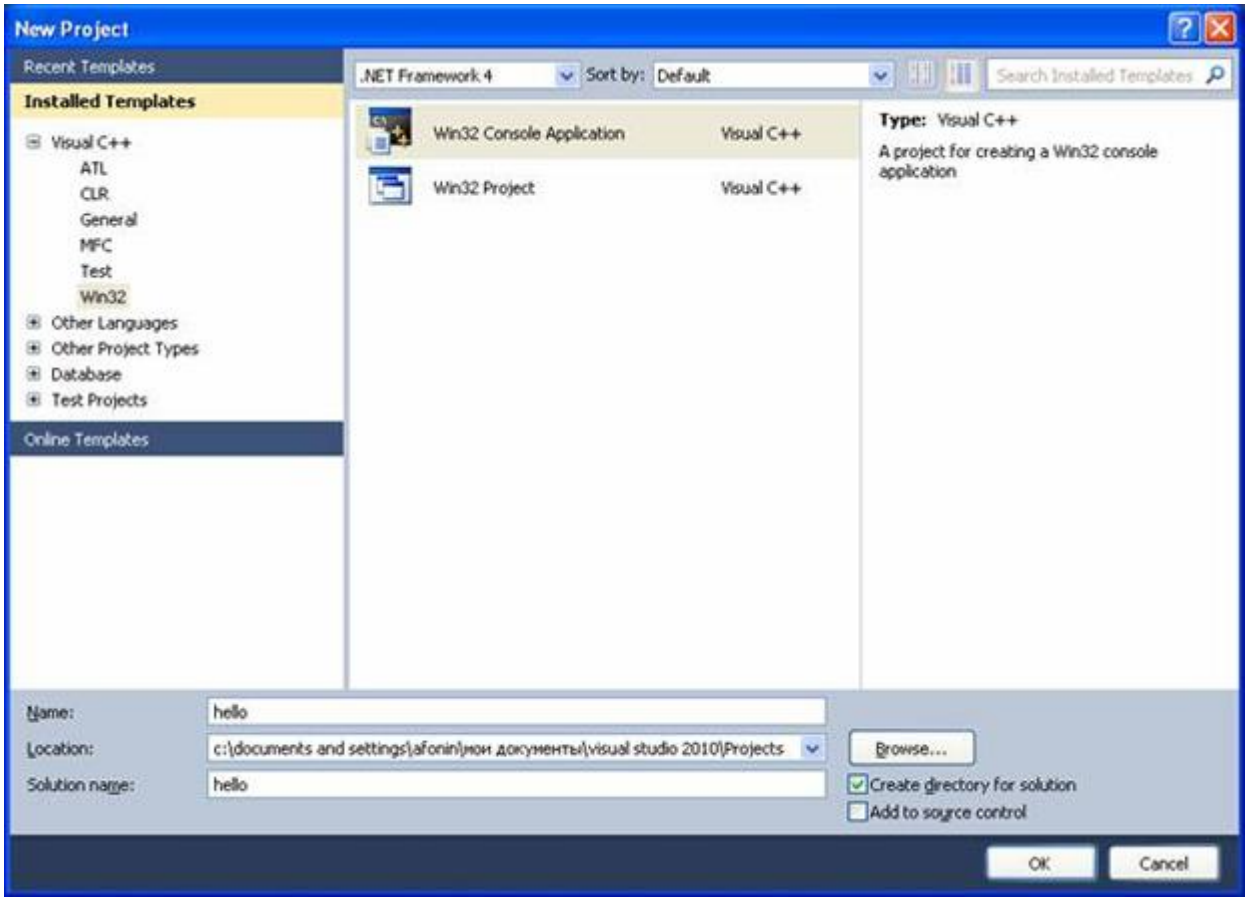

Рис. 1.4. Пример задания имени проекта

Выбор имени проекта может быть достаточно произвольным: допустимо использовать числовое *значение*, допустимо имя задавать через буквы русского алфавита.

В дальнейшем будем использовать имя, набранное с помощью букв латинского алфавита и, может быть, с добавлением цифр.

Рис. 1.5. Мастер создания приложения

На первой странице представлена *информация* о создаваемом проекте, на второй можно сделать первичные настройки проекта. После обращения к странице **Application Settings**, или после нажатия кнопки **Next** получим окно, показанное на рис. рис. 1.6.

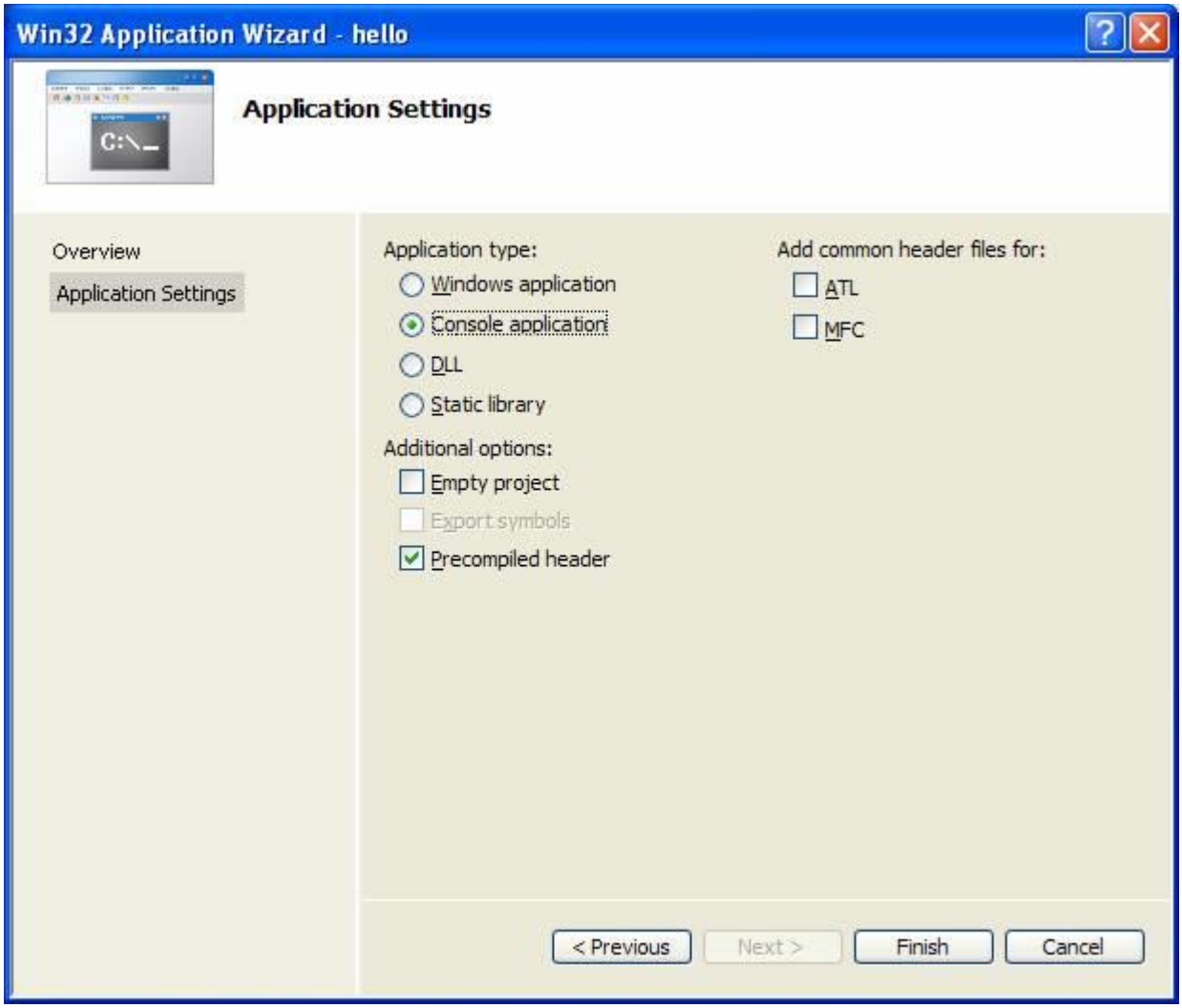

Рис. 1.6. Страница мастера настройки проекта по умолчанию

В дополнительных опциях (**Additional options**) следует поставить галочку в *поле* **Empty project** (пустой проект) и снять (убрать) галочку в *поле* **Precompiled header**. Получим экранную форму, показанную на рис. 1.7.

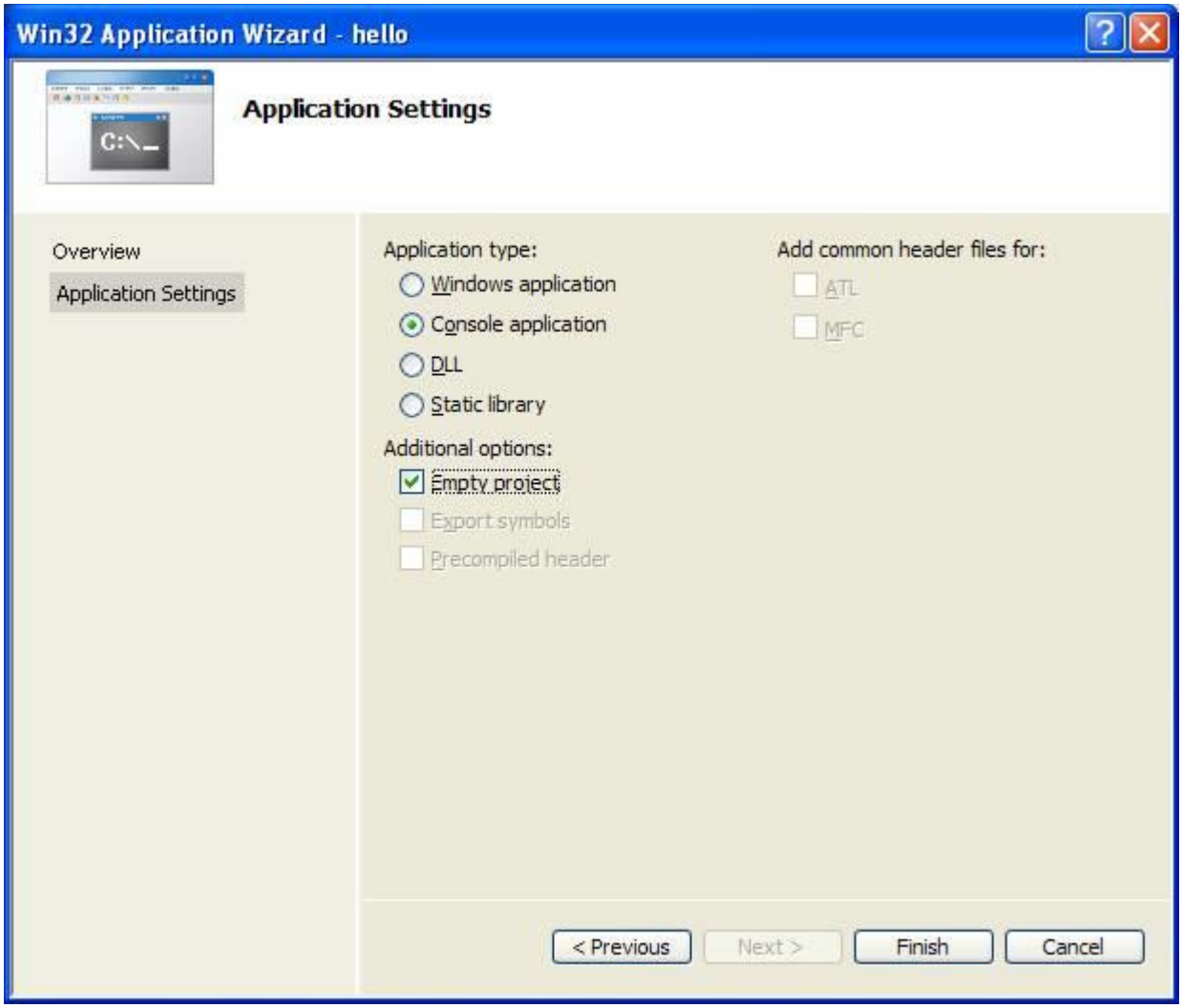

Рис. 1.7. Выполненная настройка мастера приложений

Здесь и далее будут создавать проекты *по* приведенной схеме, т.е. проекты в консольном приложении, которые должны создаваться целиком программистом (за счет выбора **Empty project**). После нажатия кнопки **Finish**, получим экранную форму, показанную на рис. 1.8, где приведена последовательность действий добавления файла для создания исходного кода к проекту. Стандартный *путь*для этого: подвести *курсор* мыши к папке **Source Files** из узла **hello** в левой части открытого проекта приложения, выбрать **Add** и**New Item** (новый элемент).

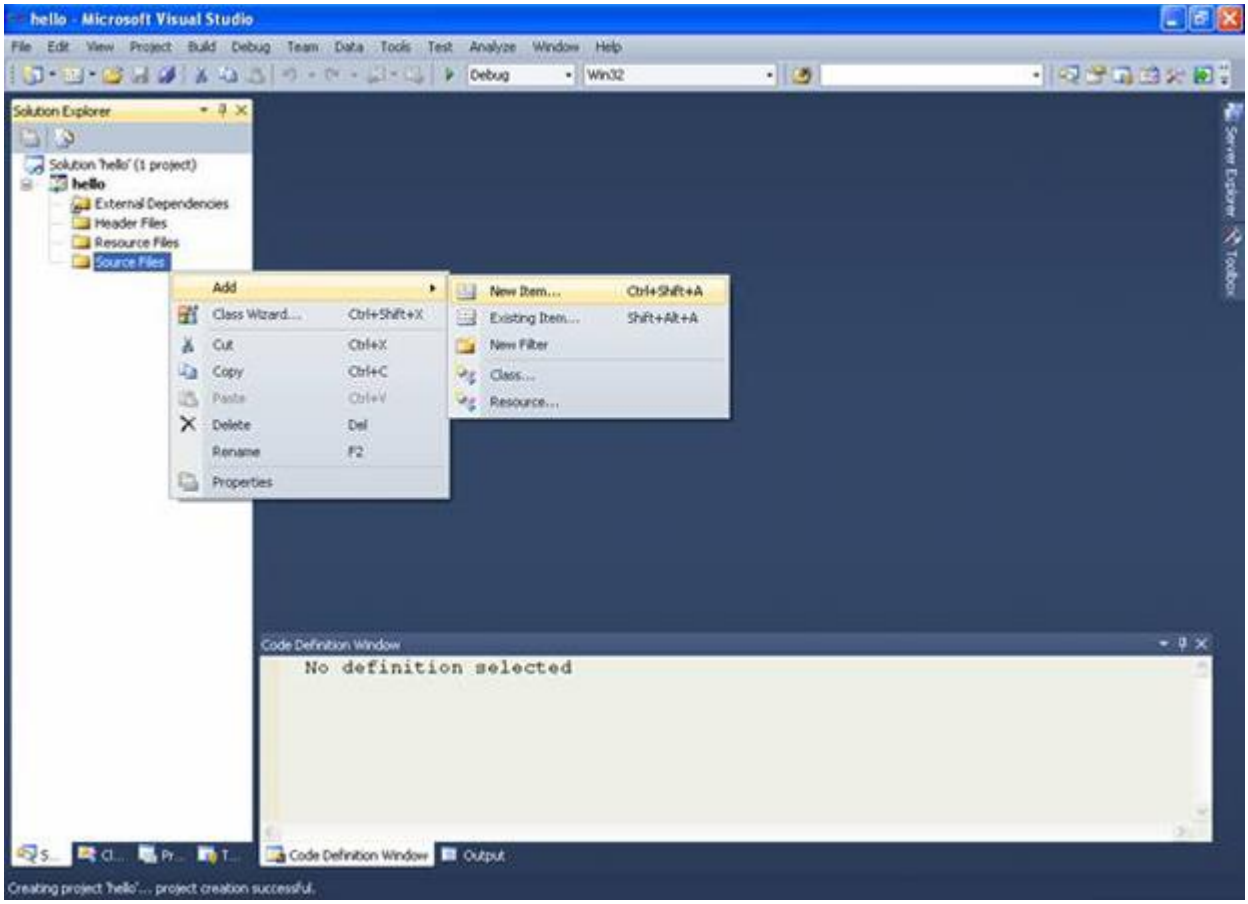

Рис. 1.8. Меню добавления нового элемента к проекту

После выбора (нажатия) **New Item** получим окно, показанное на рис. 1.9, где через *пункт меню* **Code** узла **Visual C++** выполнено обращение к центральной части панели, в которой осуществляется выбор типа файлов. В данном случае требуется обратиться к закладке **C++ File** (.cpp).

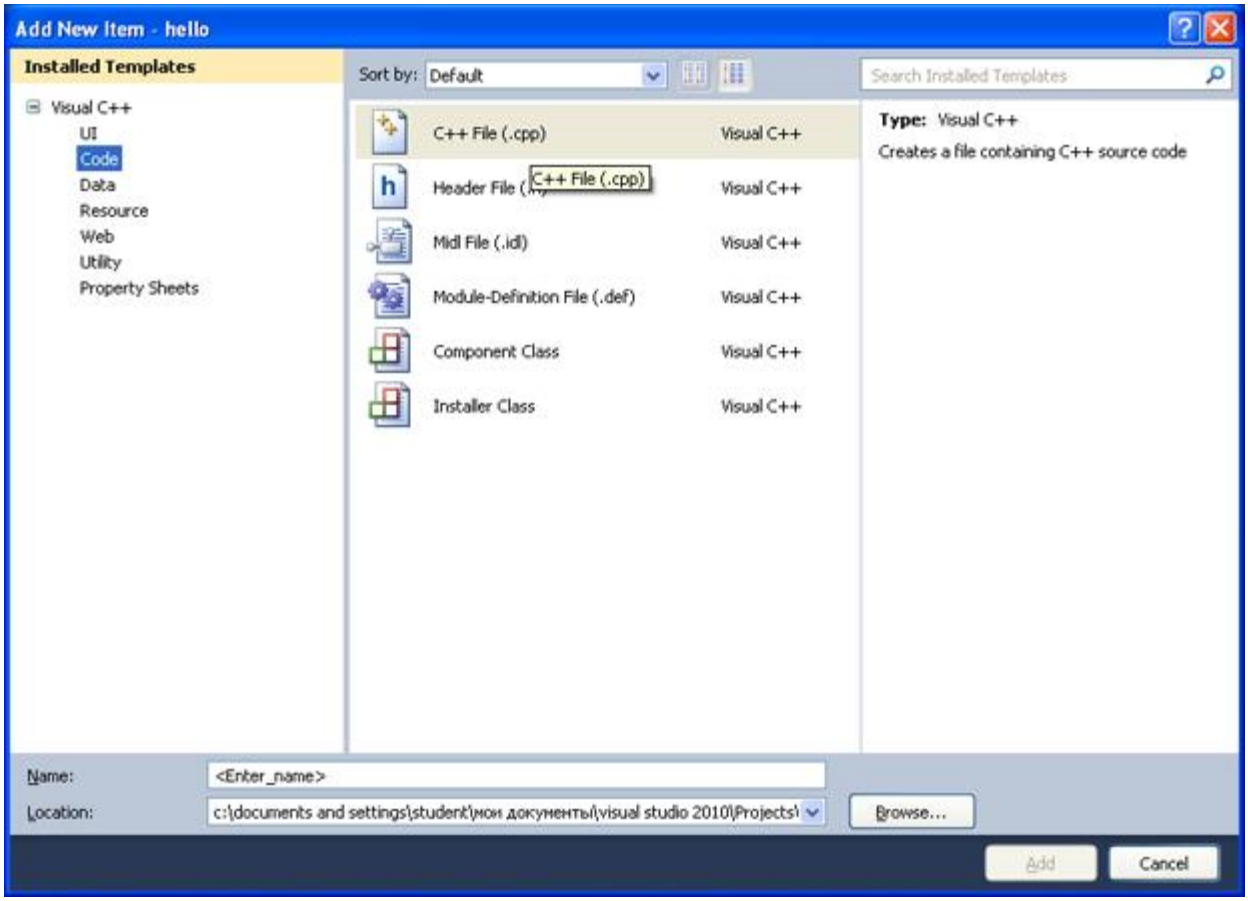

Рис. 1.9. Окно выбора типа файла для подключения к проекту

Теперь в *поле* редактора **Name** (в нижней части окна) следует задать имя нового файла и указать расширение **".c"**. Например, **main.c**.*Имя файла* может быть достаточно произвольным, но имеется негласное соглашение, что *имя файла* должно отражать его назначение и логически описывать исходный код, который в нем содержится. В проекте, состоящем из нескольких файлов, имеет смысл выделить *файл*, содержащий главную функцию программы, с которой она начнет выполняться. В данном пособии такому файлу мы будем задавать имя **main.c**, где расширение **.c** указывает на то, что этот *файл* содержит исходный код на языке **C**, и он будет транслироваться соответствующим компилятором. Программам на языке **C** принято давать расширение **.c**. После задания имени файла в *поле* редактора **Name**, получим форму, показанную на рис. 1.10.

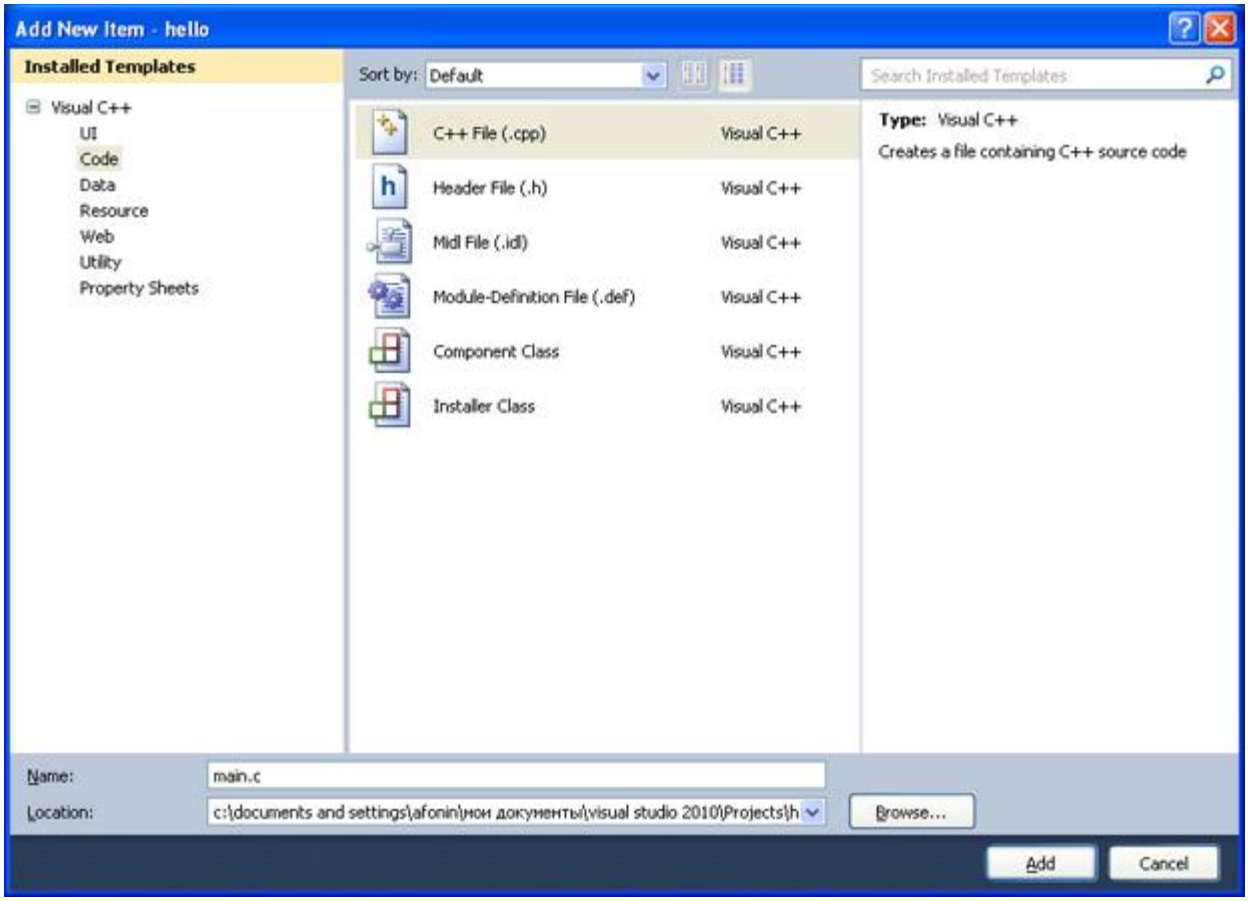

Рис. 1.10. Задание имени файла, подключаемому к проекту

Затем следует нажать кнопку **Add**. Вид среды *Visual Studio* после добавления первого файла к проекту показан на рис. 1.11. Добавленный *файл* отображается в дереве **Solution Explorer** под узлом *Source Files* (файлы с исходным кодом), и для него автоматически открывается редактор.

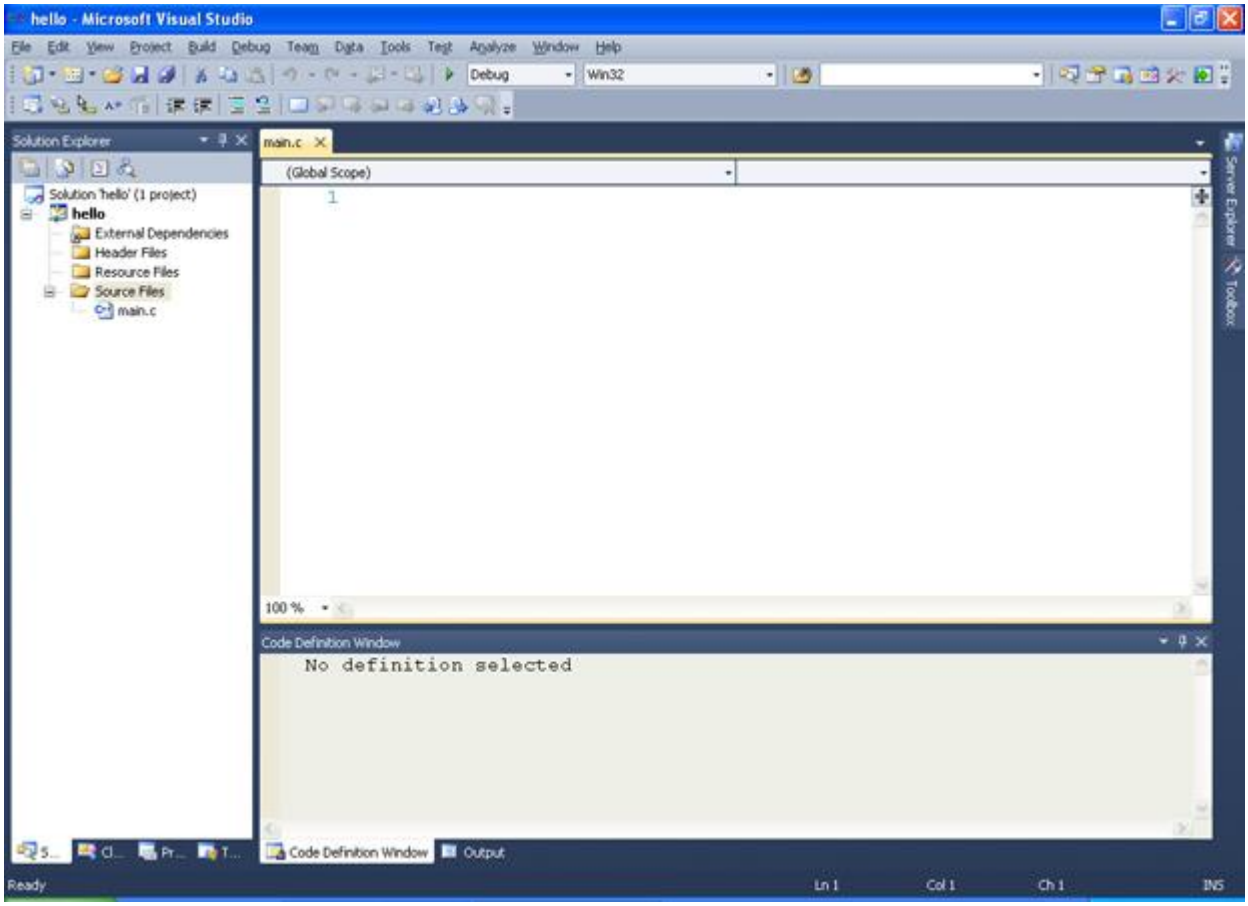

Рис. 1.11. Подключение файла проекта

На рис. 1.11 в левой панели в папке **Solution Explorer** отображаются файлы, включенные в проект в папках. Приведем описание.

Папка **Source Files** предназначена для файлов с исходным кодом. В этой папке отображаются файлы с расширением **.c**.

Папка **Header Files** содержит заголовочные файлы с расширением **.h**.

Папка **Resource Files** содержит *файлы ресурсов*, например изображения и т. д.

Папка **External Dependencies** отображает файлы, не добавленные явно в проект, но использующиеся в файлах исходного кода, например включенные при помощи директивы #include. Обычно в папке **External Dependencies** присутствуют заголовочные файлы стандартной библиотеки, использующиеся в проекте.

Следующий шаг состоит в настройке проекта. Для этого в *меню* **Project** главного *меню* следует выбрать **hello Properties** (или с помощью последовательного нажатия клавиш Alt+F7). Пример обращения к этому пункту *меню* показан на рис. 1.12.

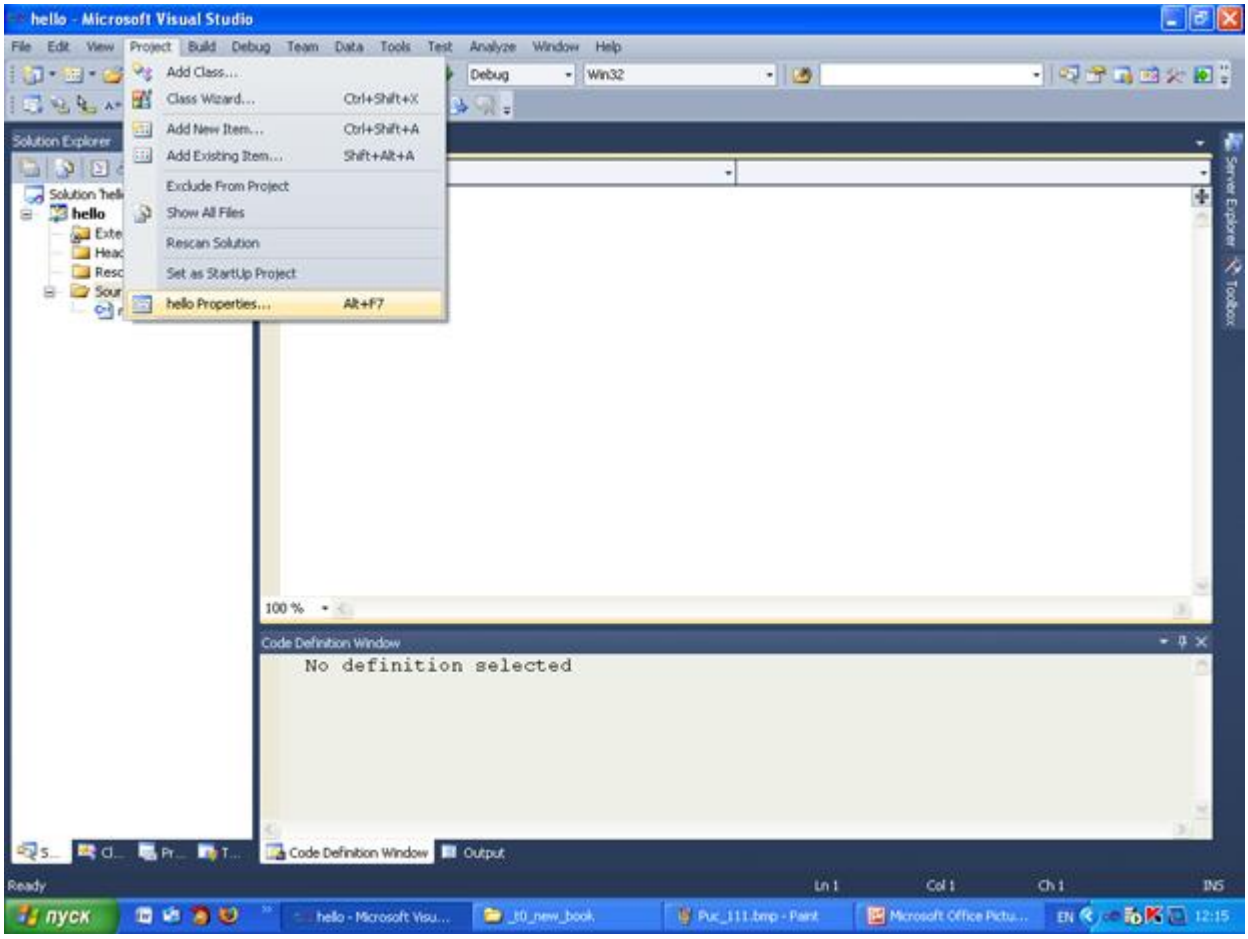

Рис. 1.12. Обращение к странице свойств проекта

После того как произойдет открытие окна свойств проекта, следует обратиться (с левой стороны) к **Configuration Properties**. Появится ниспадающий *список*, который показан на рис. 1.13. Выполнить обращение к узлу *General*, и через него в правой панели выбрать **Character Set**, где установить свойство **Use Multi-byte Character Set.** Настройка **Character Set** (набор символов) позволяет выбрать, какая *кодировка* символов – *ANSI* или UNICODE – будет использована при компиляции программы. Для совместимости со стандартом C89 мы выбираем **Use Multi-Byte Character Set**. Это позволяет использовать многие привычные функции, например, функции *по* выводу информации на *консоль*.

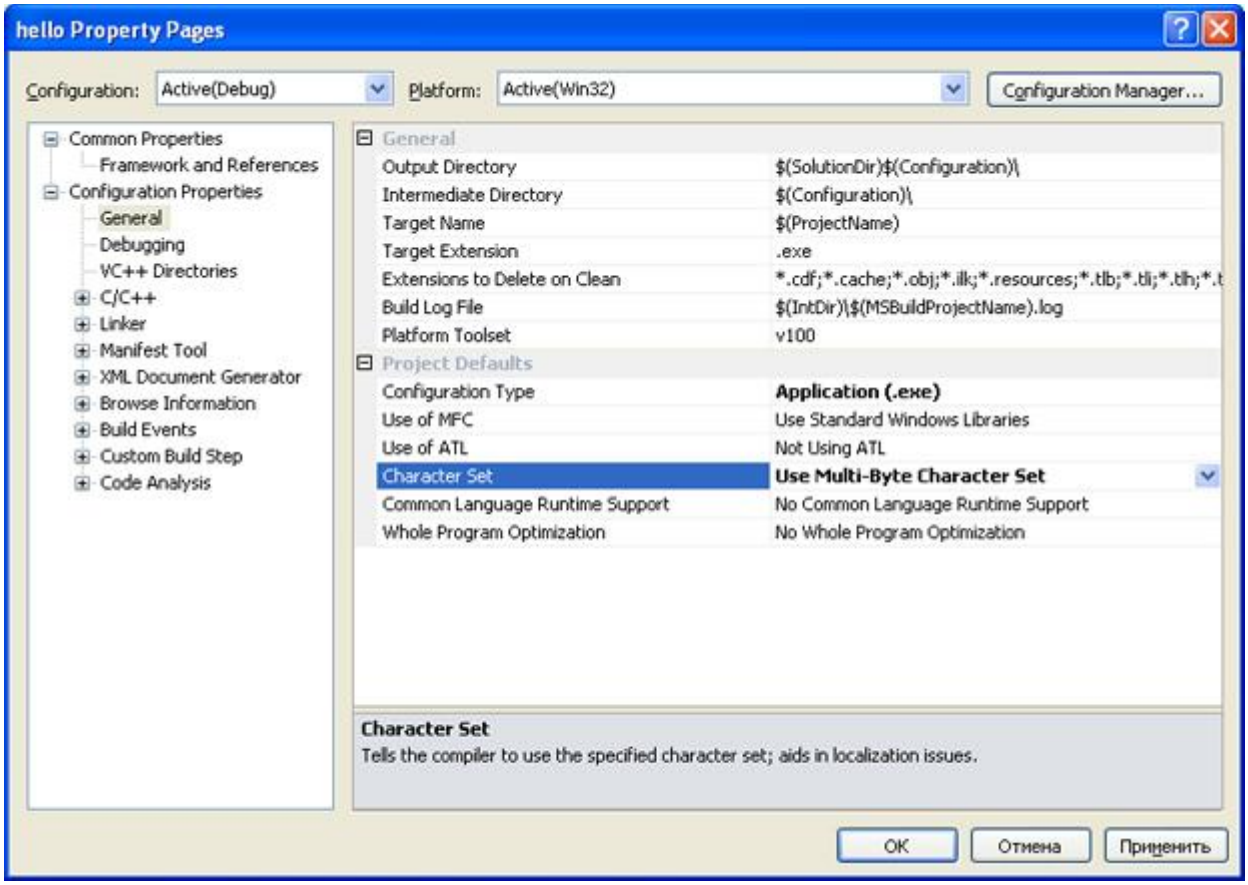

Рис. 1.13. Меню списка свойств проекта

После сделанного выбора, показанного на рис. 1.13, следует нажать кнопку Применить. Затем следует выбрать узел **C/C++** и в ниспадающем *меню* выбрать *пункт* **Code Generation**, через который следует обратиться в правой части панели к закладке **Enable C++ Exceptions**, для которой установить **No** (запрещение исключений C++). Результат установки выбранного свойства показан на рис. 1.14. После произведенного выбора нажать кнопку **Применить**.

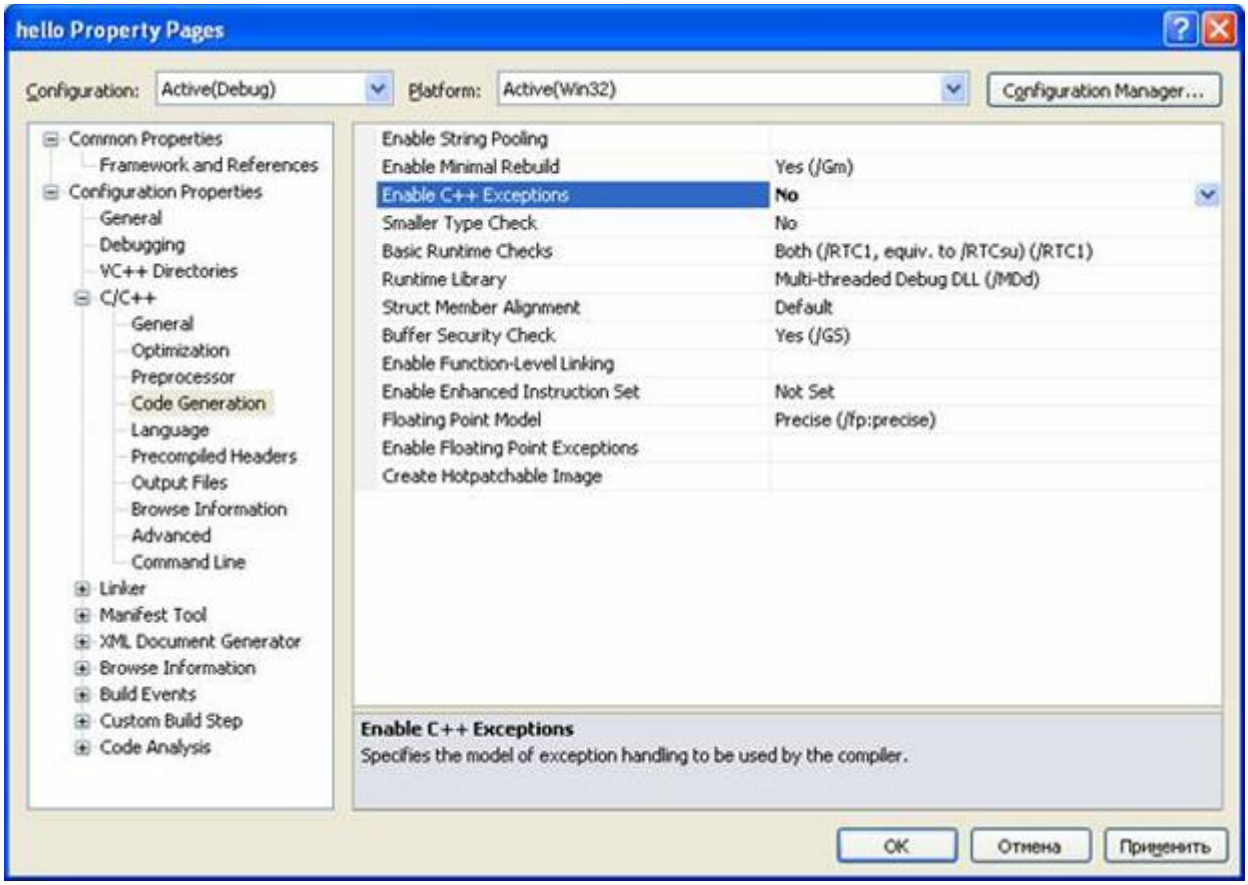

Рис. 1.14. Страница свойств для запрещения исключений C++

Далее в ниспадающем *меню* узла **C/C++** необходимо выбрать *пункт* **Language** и через него обратиться в правую часть панели, где установить следующие свойства: свойство **Disable Language Extensions** (дополнительные языковые расширения фирмы Microsoft) в **Yes (/Za)**, свойство **Treat wchar_t as Built-in Type** (рассматривать тип wchar_t как встроенный тип) установить в **No (/Zc:wchar_t–)**, свойство **Force Conformance in For Loop Scope** (соответствие стандарту определения локальных переменных в операторе *цикла* for) установить в Yes(/Zc:forScope), свойство **Enable Run-Time Type Info** (разрешить информацию о типах во *время выполнения*) установить в **No (/GR–)**, свойство **Open MP Support** (разрешить расширение *Open* MP – используется при написании программ для многопроцессорных систем) установить в **No(/openmp–)**.

Результат выполнения этих действий показан на рис. 1.15.

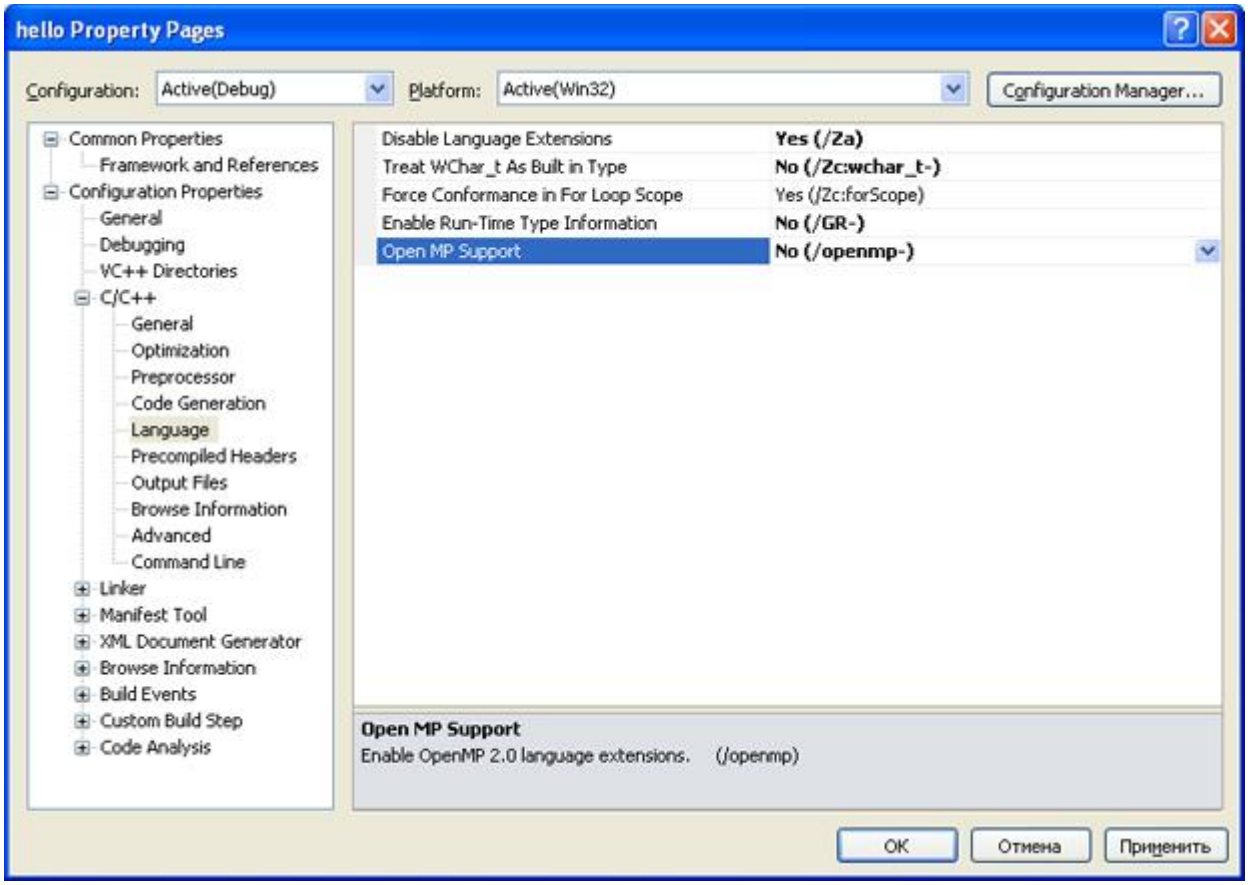

Рис. 1.15. Страница свойств закладки Language

После выполнения указанных действий следует нажать клавишу **Применить**. Далее в ниспадающем списке узла **C/C++** следует выбрать *пункт* **Advanced** и в правой панели изменить свойство **Compile As** в свойство компиляции языка **C**, т.е. **Compile as C Code (/TC)**. Результат установки компилятора языка **C** показан на рис. 1.16.

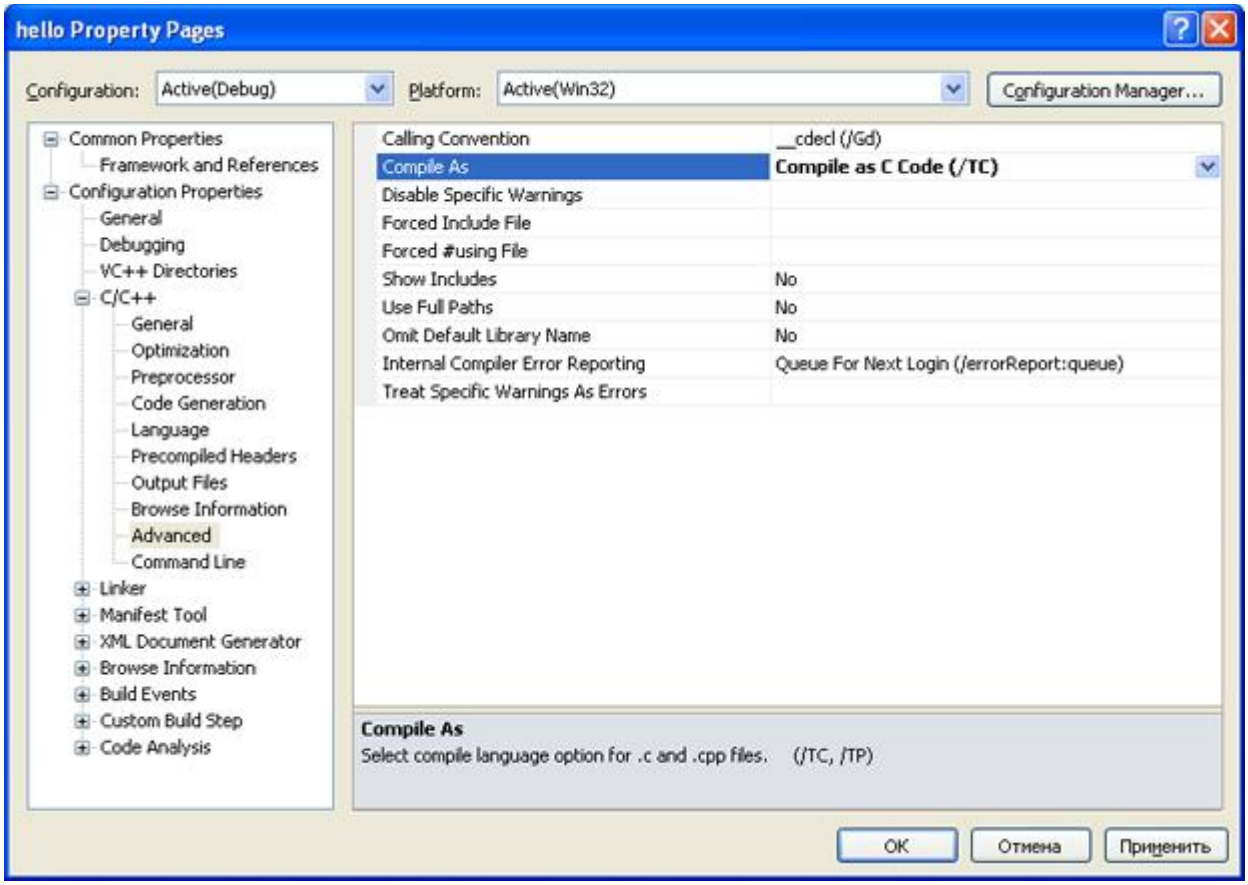

Рис. 1.16. Результат выбора режима компиляции языка C

После нажатия клавиш **Применить** и **ОК** сначала откроется подготовленный проект с пустым полем редактора кода, в котором можно начать писать программы. В этом редакторе наберем программу, выводящую традиционное приветствие "Hello World". Для компиляции созданной программы можно обратиться в *меню* **Build**, или, например, нажать клавишу F6. В случае успешной компиляции получим следующую экранную форму, показанную на рис. 1.17.

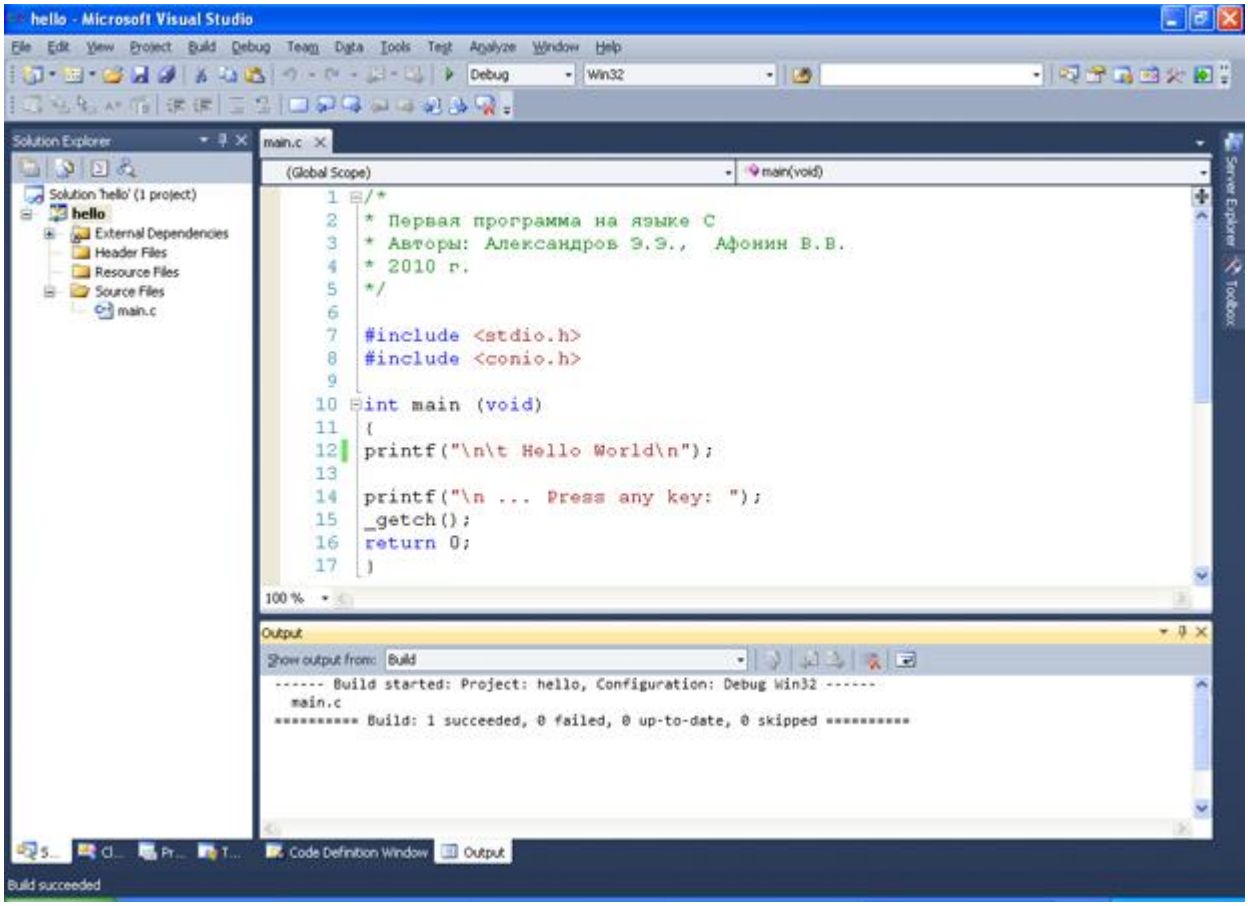

Рис. 1.17. Успешно откомпилированная первая программа на языке C

Для приведенного кода программы *запуск* на ее *исполнение* из окна редактора в *Visual Studio* 2010 можно нажать клавишу **F5**. рис. 1.18 показан результат исполнения первой программы.

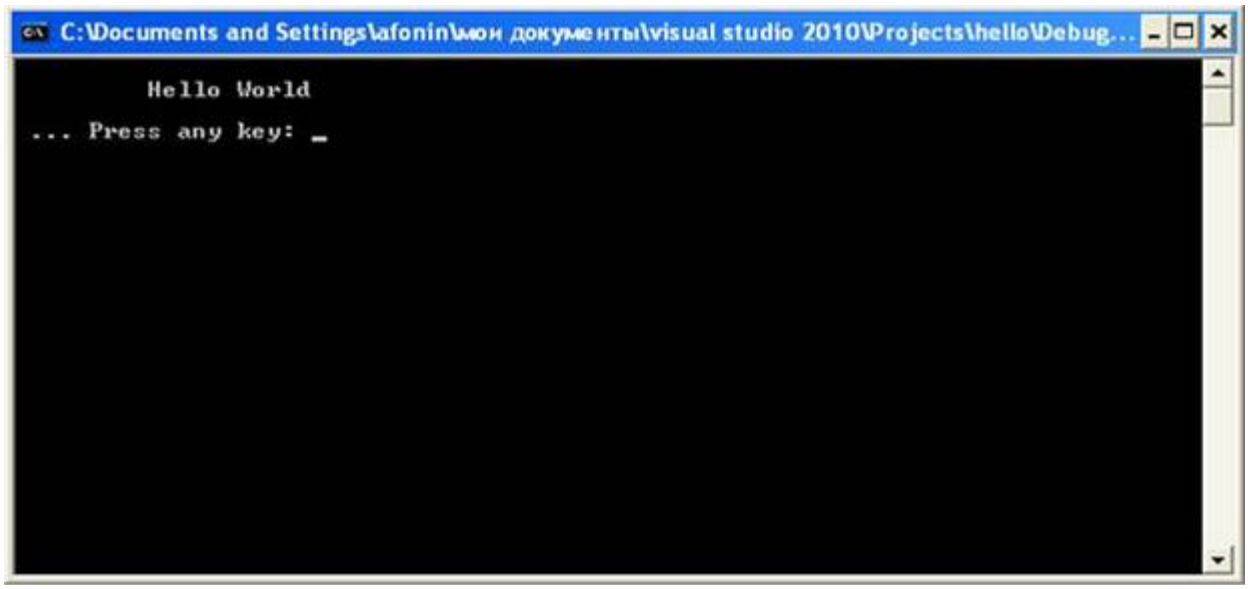

Рис. 1.18. Консольный вывод первой программы на языке C

Произведем разбор первой программы. Во-первых, надо отметить, что в языке **C** нет стандартных инструкций (операторов) для вывода сообщений на *консоль* (окно пользователя). В языке **C** предусматриваются специальные библиотечные файлы, в которых имеются функции для этих целей. В приведенной программе используется заголовочный *файл* с именем stdio.h (*стандартный ввод–вывод*), который должен быть включен в начало программы. Для вывода сообщения на *консоль* используется *функция* printf(). Для работы с консолью включен также заголовочный *файл* conio.h., который поддерживает функцию _getch(), которая извлекает символ из потока ввода, т. е. она предназначенная для приема сообщения о нажатии какой-либо (почти любой) клавиши на клавиатуре. С другими компиляторами, возможно, потребуется getch(), т.е. без префиксного нижнего подчеркивания. Строка программы

```
int main (void)
```

сообщает системе, что именем программы является main() – главная *функция*, и что она возвращает *целое число*, о чем указывает аббревиатура **"int"**. Имя main() – это специальное имя, которое указывает, где *программа* должна начать выполнение. Наличие круглых скобок после слова main() свидетельствует о том, что это *имя функции*. Если содержимое круглых скобок отсутствует или в них содержится *служебное слово* **void**, то это означает, что в функцию main() не передается никаких аргументов. *Тело функции* main() ограничено парой фигурных скобок. Все утверждения программы, заключенные в *фигурные скобки*, будут относиться к функции main().

В теле функции main() имеются еще три функции. Во-первых, функции printf() находятся в библиотеке компилятора языка **C**, и они печатают или отображают те аргументы, которые были подставлены вместо параметров. Символ "\n" составляет единый символ *newline* (новая строка), т.е. с помощью этого символа осуществляется перевод на новую строку. Символ "\t" осуществляет табуляцию, т.е. начало вывода результатов программы с отступом вправо.

Функция без параметров _getch() извлекает символ из потока ввода (т.е. ожидает нажатия почти любой клавиши). С другими компиляторами, возможно, потребуется getch(), т.е. без префиксного нижнего подчеркивания.

Последнее утверждение в первой программе

```
return 0;
```

указывает на то, что выполнение функции main() закончено и что в систему возвращается *значение* 0 (*целое число*). Нуль используется в соответствии с соглашением об индикации успешного завершения программы.

В завершение следует отметить, что все действия в программе завершаются символом точки с запятой.

Если в ходе компиляции возникли ошибки, то Visual Studio сообщит о том, что найдены ошибки и предложит запустить последний успешно построенный вариант.
Разумеется, нужно отказаться и просмотреть какие именно ошибки были найдены компилятором. Сообщение может не появиться, а сразу отобразится окно со списком ошибок. Если же список ошибок не появился, а запустился ранее собранный проект, сначала в меню View («Вид») выберите Error list («Список ошибок»), а затем в меню Build («Построение») выберите Build Solution («Построить решение»). Для каждой ошибки

отображается описание, имя проекта, имя файла, номер строки и столбца, где встретилась эта ошибка в указанном файле. Если дважды щелкнуть левой кнопкой мыши по ошибке, то курсор в файле с исходным кодом будет помещен в указанную строку и столбец. Ошибка находится не всегда в том месте, где указывает ее среда разработки. Поэтому важно уметь анализировать и правильно понимать описание ошибки.

Все файлы проекта сохраняются в той папке, которая сформировалась после указания в *поле Location* имени проекта (hello). На рис. 1.19 показаны папки и файлы проекта *Visual Studio* 2010..

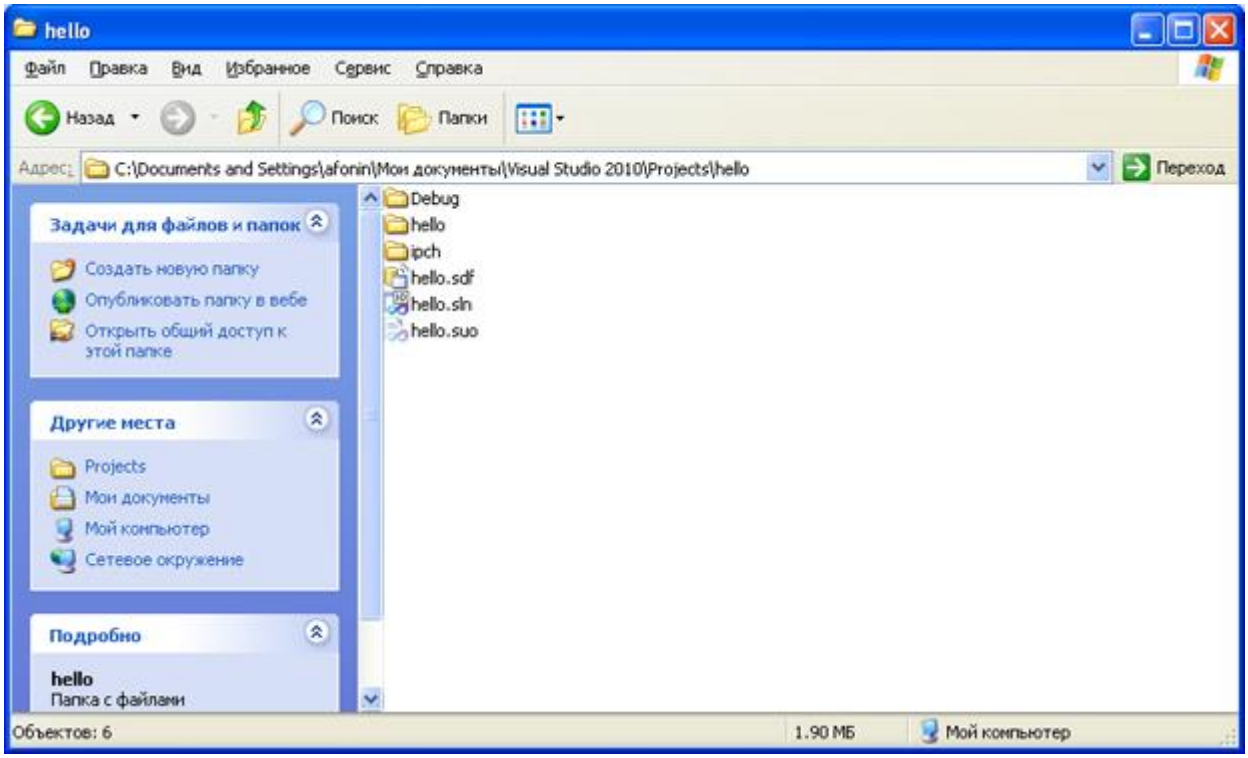

Рис. 1.19. Файлы и папки созданного проекта

На рис. 1.19 файлы с полученными расширениями означают:

hello.sln – *файл* решения для созданной программы. Он содержит информацию о том, какие проекты входят в данное решение. Обычно, эти проекты расположены в отдельных подкаталогах. Например, наш проект находится в подкаталоге hello;

hello.suo – *файл* настроек среды *Visual Studio* при работе с решением, включает информацию об открытых окнах, их расположении и прочих пользовательских параметрах.

hello.sdf – *файл* содержащий вспомогательную информацию о проекте, который используется инструментами анализа кода *Visual Studio*, такими как IntelliSense для отображения подсказок об именах и т.д.

Файлы папки **Debug** показаны на рис. 1.20.

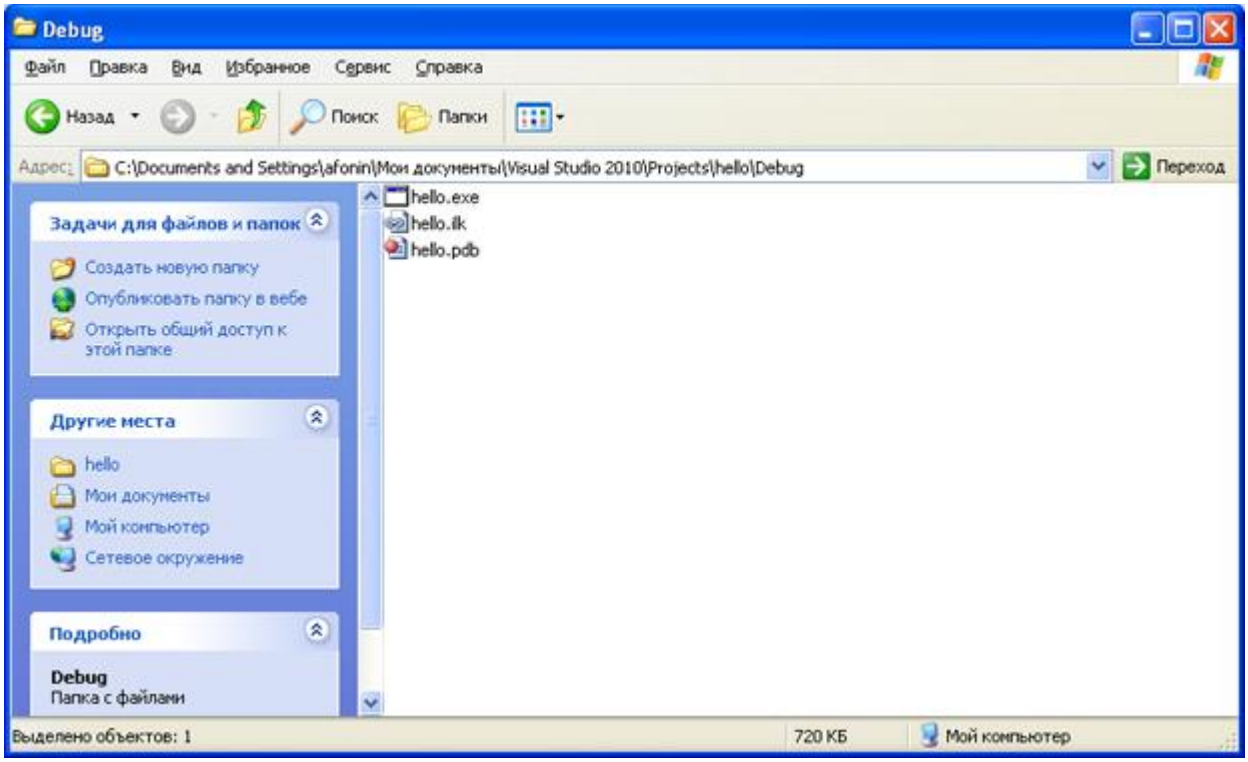

Рис. 1.20. Файлы папки Debug

Рассмотрим файлы в соответствии с рис. 1.20.

hello.exe – исполняемый *файл* проекта;

hello.ilk – *файл* "*incremental* linker", используемый компоновщиком для ускорения процесса компоновки;

hello.pdb – отладочная *информация*/*информация* об именах в исполняемых файлах, используемая отладчиком.

Файлы папки **hello** показаны на рис. 1.21.

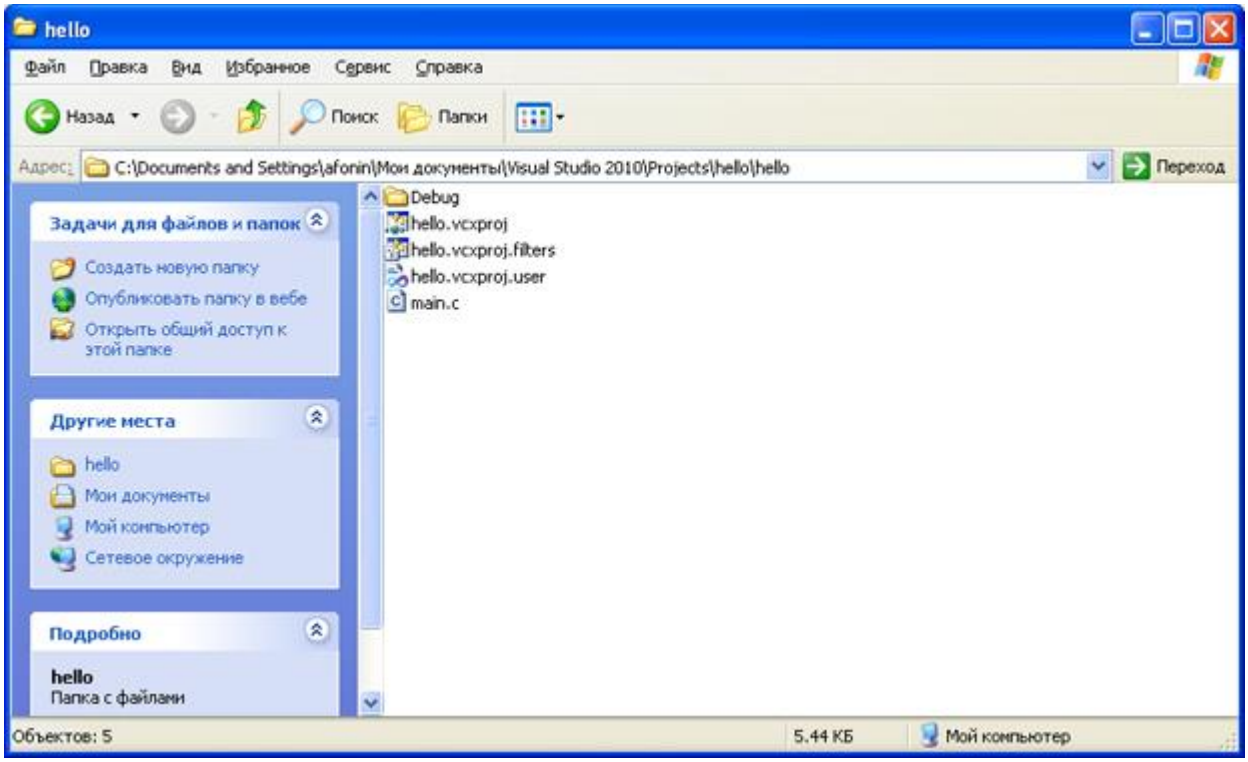

Рис. 1.21. Содержимое папки hello

Характеристика содержимого папки **hello**:

main.c – *файл* исходного программного кода,

hello.vcxproj – *файл* проекта,

hello.vcxproj.user – *файл* пользовательских настроек, связанных с проектом,

hello.vcxproj.filters – *файл* с описанием фильтров, используемых *Visual Studio Solution Explorer* для организации и отображения файлов с исходным кодом.

Задание

1. Создайте проект консольного приложения и настройте его так, как показано в теоретической части.

2. Добавьте в проект Source файл скопируйте в него следующий текст программы:

```
#include “lab2.h”
int main (void)
{
        return 0;
}
```

3. Попробуйте произвести компиляцию и построение решения. Опишите, что произошло и почему?

4. Добавьте в проект Header файл с именем lab2.h и скопируйте в него следующий текст:

```
#include <stdio.h>
#include <conio.h>
```

5. Напишите программу, которая выводила бы на консоль название факультета, где учитесь, номер группы, свою фамилию, имя и отчество в разных строках дисплея (консоли) с помощью одной функции printf() (см. пример **hello** в теоретической части).

6. Вывод выполните с помощью нескольких функций printf() (количество функций должно соответствовать каждой порции информации).

7. Удалите ключевое слово void для функции main(). Опишите, что произошло и почему?

8. Закомментируйте строку, содержащую функцию main. Опишите, что произошло и почему?

9. (дополнительное задание, не обязательное для выполнения) Изучите теорию из Приложения (см. ниже). Вставьте в функцию main код из Примера 2. Скомпилируйте и выполните Debag версию вашего приложения. Скомпилируйте и выполните Release версию вашего приложения. Опишите, чем отличается результаты выполнения данных версий и почему?

   Примечание. *Вывод* требуемой информации осуществляется с помощью букв латинского алфавита. Комментарии в программе могут быть сделаны после символа "//" или внутри комбинации символов "/* */".

Контрольные вопросы

1. Какие компиляторы языка **C** вам известны?
2. Какое имя имеет исполняемый файл созданного проекта?
3. Объясните назначение заголовочных файлов stdio.h, conio.h.
4. Как будет работать программа без заголовочного файла conio.h?
5. В каком месте программы находится точка ее входа?
6. Как осуществляется табуляция строки на консоли и на сколько позиций выполняется отступ от левого края?
7. Какое значение имеет главная функция проекта main() в программах на языке **C**?

Приложение

Любой проект в *Visual Studio* 2010 включает несколько самостоятельных конфигураций для компиляции разных версий программы. Конфигурацией называется набор параметром компилятора, компоновщика и библиотекаря, используемый при построении проекта. *По* умолчанию, при создании проекта, среда *Visual Studio* 2010 автоматически включает в него две конфигурации: *Debug*(отладочная) и *Release* (финальная). Как следует из их названий, отладочная *конфигурация* используется в процессе написания программы, ее тестовых запусков для обнаружения и исправления ошибок; в то время как финальная *конфигурация* используется для построения конечной версии продукта, передаваемого заказчику для промышленного использования.

При создании проекта настройки отладочной (*Debug*) и финальной (*Release*) конфигураций устанавливаются в значения *по*умолчанию. С этими настройками выполняются следующие действия:

- Debug (отладочная) конфигурация проекта компилируется с включением полной символьной отладочной информации и выключенной оптимизацией. Оптимизация кода затрудняет процесс отладки, так как усложняет или даже полностью изменяет отношение между строками исходного кода программы и сгенерированными машинными инструкциями. Такая отладочная информация используется отладчиком Visual Studio 2010 для отображения значений переменных, определения текущей выполняемой строки программы, отображения стека вызовов и так далее, то есть для поддержки стандартных действий, выполняемых при отладке программы.
- Release (финальная) конфигурация проекта не содержит никакой отладочной информации и подвергается полной оптимизации. Без отладочной информации процесс отладки программы очень затруднен. Однако, при необходимости, отладочная информация может быть создана и для финальной версии программы и записана в отдельный файл с расширением .pdb. Файлы отладочной информации .pdb могут оказаться очень полезными, если позднее возникнет необходимость в отладке финальной версии программы, например, при обнаружении критических ошибок в процессе ее эксплуатации на компьютерах заказчика. Файлы .pdb обычно заказчику не передаются, а сохраняются у разработчиков.

Переключение между конфигурациями можно осуществлять из панели инструментов или при помощи окна *Configuration Manager*(*менеджер* конфигураций).

Для быстрого переключения конфигурации, используемой для компиляции проекта, используется стандартная *панель инструментов*(рис. 24.1).

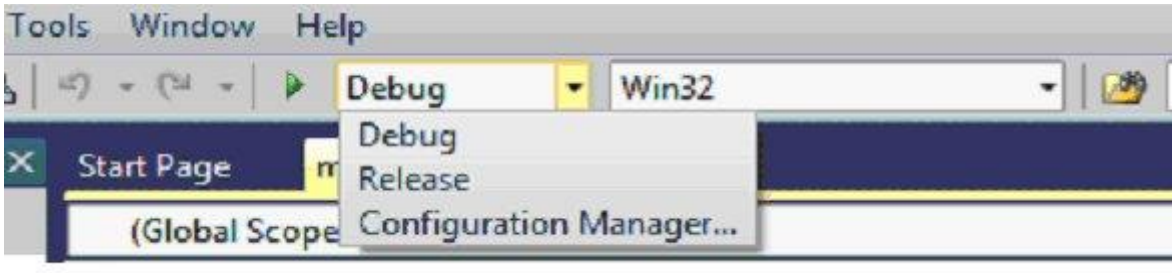

Рис. 24.1. Переключение конфигураций из панели инструментов

Программист может в любой момент изменить настройки конфигурации проекта или, при необходимости, создать новую конфигурацию. Эти действия выполняются в окне свойств проекта. Настройки свойств проекта применяются к текущей выбранной конфигурации. Можно указать одну из созданных конфигураций проекта, или выбрать *All* configurations, в этом случае настройки будут применены ко всем конфигурациям одновременно. Рассмотрим основные отличия в настройках проекта для отладочной и финальной конфигураций. На рис. 24.2 изображено окно свойств проекта со страницей настроек оптимизации (*Optimization*) для отладочной конфигурации проекта. Видно, что для отладочной конфигурации *оптимизация* генерируемого машинного кода выключена (Disabled).

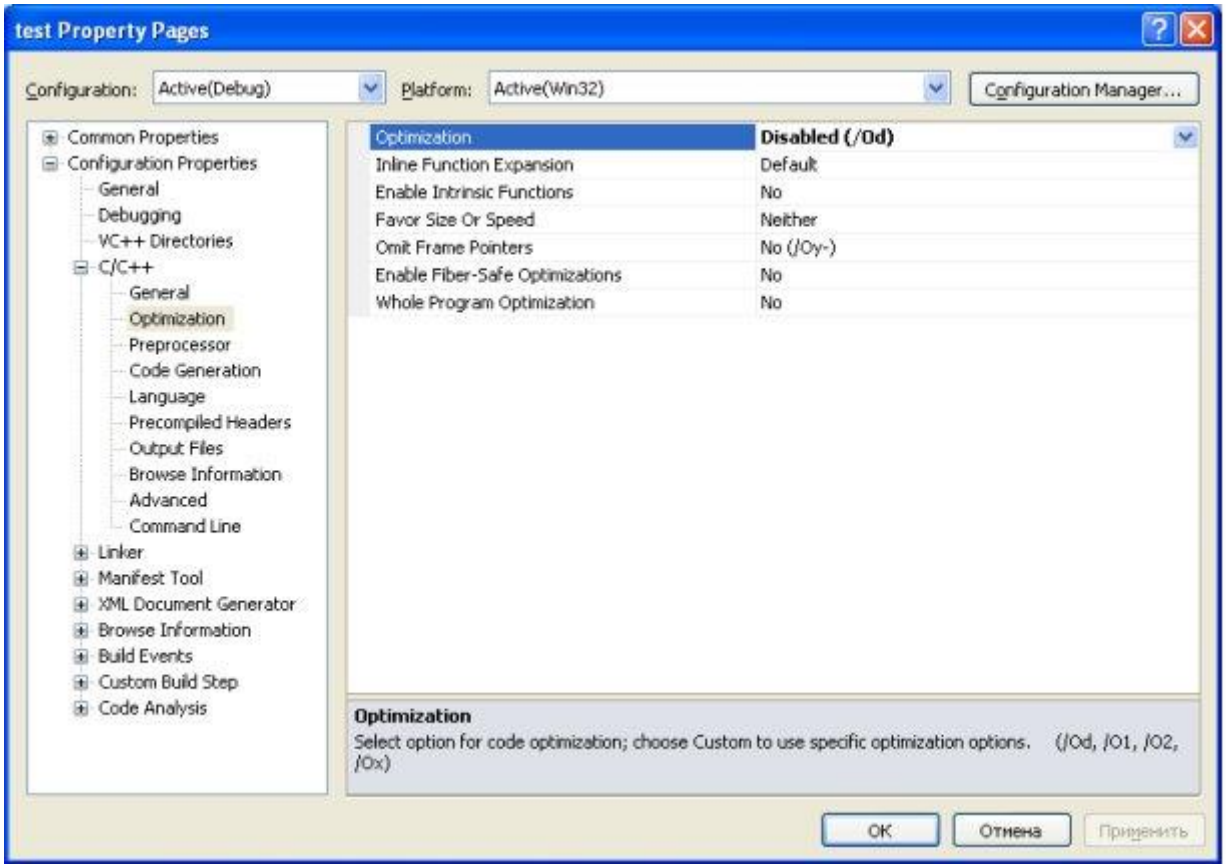

Рис. 24.2. Страница свойств Optimization для отладочной конфигурации

Для финальной версии проекта *по* умолчанию включена *оптимизация по* скорости выполнения программы (Optimize *Speed*). На рис. 24.3 показана страница с выбранными настройками финальной конфигурации.

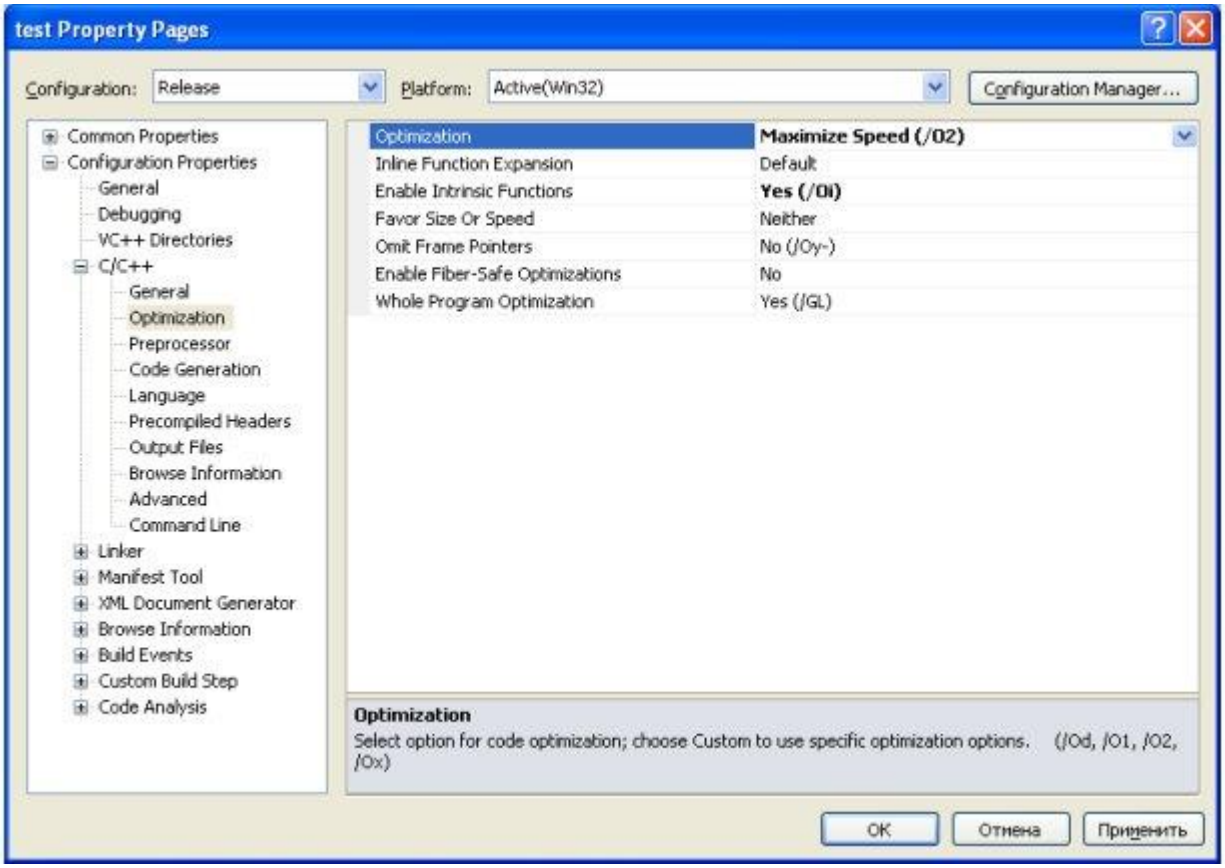

Рис. 24.3. Страница свойств Optimization для финальной конфигурации

Кроме того, можно также выбрать следующие параметры оптимизации – генерировать максимально компактный код (*Minimize Size*) и полная *оптимизация* (*Full Optimization*), которая включает максимальные настройки оптимизации. На рис. 24.4 показан *список* свойств закладки *Optimization*.

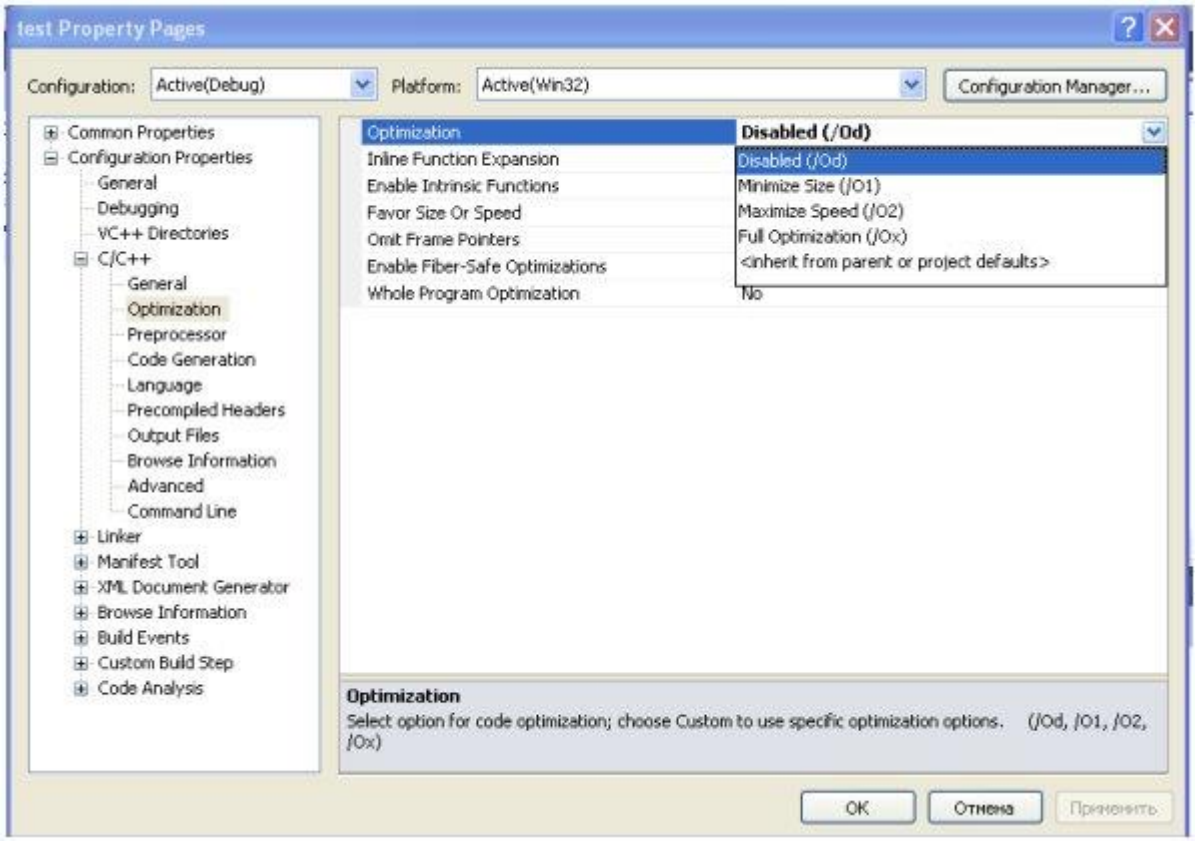

Рис. 24.4. Список свойств закладки Optimization

На странице свойств генерации кода (*Code Generation*) можно указать версию стандартной библиотеки языка C, которая будет использована при компиляции и компоновке проекта – настройка Runtime *Library*. Для отладочной конфигурации *по* умолчанию используется настройка Multithreaded *Debug DLL* (многопоточная отладочная версия стандартной библиотеки, размещенной в динамически загружаемом модуле *DLL*). Эта версия библиотеки содержит отладочную информацию. Она также поддерживает дополнительные проверки времени выполнения, что позволяет, с одной стороны выполнять функции стандартной библиотеки под управлением отладчика, а с другой стороны – обнаруживать на раннем этапе трудно обнаруживаемые проблемы, такие как *выход* за границы массивов, неправильную работу с динамически распределяемой памятью и некоторые другие. Из-за наличия этих дополнительных проверок отладочная версия библиотеки выполняется медленнее финальной.

Для финальной версии проекта *по* умолчанию используется настройка Multithreaded *DLL* (финальная версия стандартной библиотеки без отладочной информации, размещенной в динамически загружаемом модуле *DLL*). Важным моментом является то, что для запуска финальной версии программы при компиляции ее с такими настройками, *модуль DLL* стандартной библиотеки должен присутствовать в системе. Он устанавливается либо при установки *Visual Studio* 2010, либо при помощи отдельного инсталляционного пакета Microsoft *Visual* C++ 2010 Redistributable *Package* (*x86*). Если же библиотека *DLL* в системе не установлена, то скомпилированная *программа* не будет запущена.

Для исключения зависимости от отдельной *DLL* стандартной библиотеки *значение* настройки Runtime *Library* нужно установить в Multithreaded (многопоточная версия стандартной библиотеки). В этом случае весь необходимый

функционал будет включен непосредственно в результирующий .exe *файл*, который может быть запущен и исполнен независимо от того, были ли установлены файлы *DLL* стандартной библиотеки или нет.

На рис. 24.5 показана страница свойств закладки *Code Generation* для отладочной конфигурации.

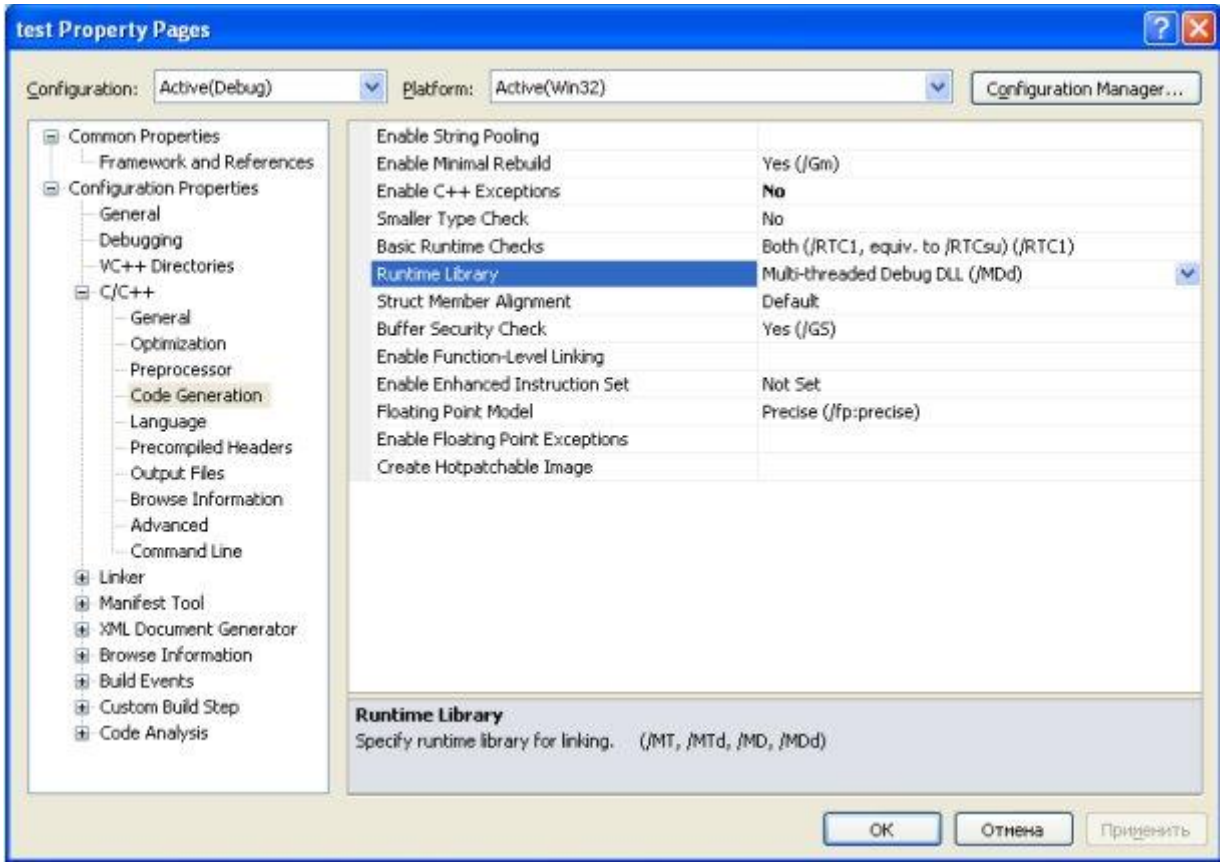

Рис. 24.5. Страница свойств Code Generation для отладочной конфигурации

На рис. 24.6 показана страница свойств закладки *Code Generation* для финальной конфигурации.

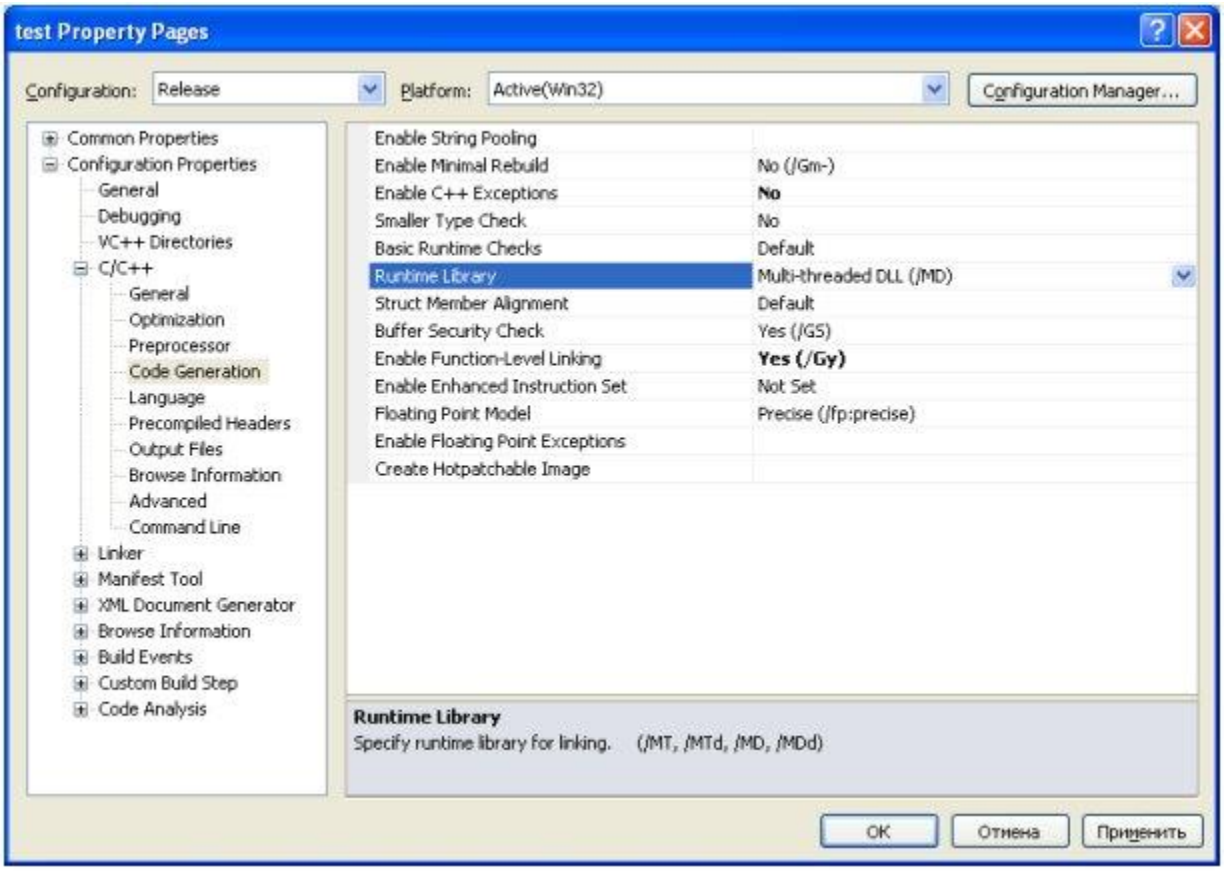

Рис. 24.6. Страница свойств Code Generation для финальной конфигурации

Отладочная и финальная версия программы компилируются также с различными символами препроцессора . Для отладочной версии объявляется символ препроцессора _DEBUG, а для финальной версии – символ NDEBUG. Это позволяет использовать директивы препроцессора для условной компиляции программы, включая или исключая некоторую функциональность в отладочную или финальную версию программы. Обычно это используется для включения дополнительных проверок и отладочного вывода в отладочную версию программы. Для финальной версии такие проверки и *вывод* не нужны, поэтому они в нее не включаются. Например, в следующем фрагменте программы *значение* переменной *res* будет выведено на экран только в отладочной версии.

Пример 2.

```
int a, b;
int res;
a = 10;
b = 20;
res = a + b;
 #ifdef _DEBUG
printf ("res = %d", res);
 #endif
```

На рис. 24.7 представлена страница свойств Preprocessor для отладочной конфигурации. На рис. 24.8 представлена страница свойств Preprocessor для финальной конфигурации.

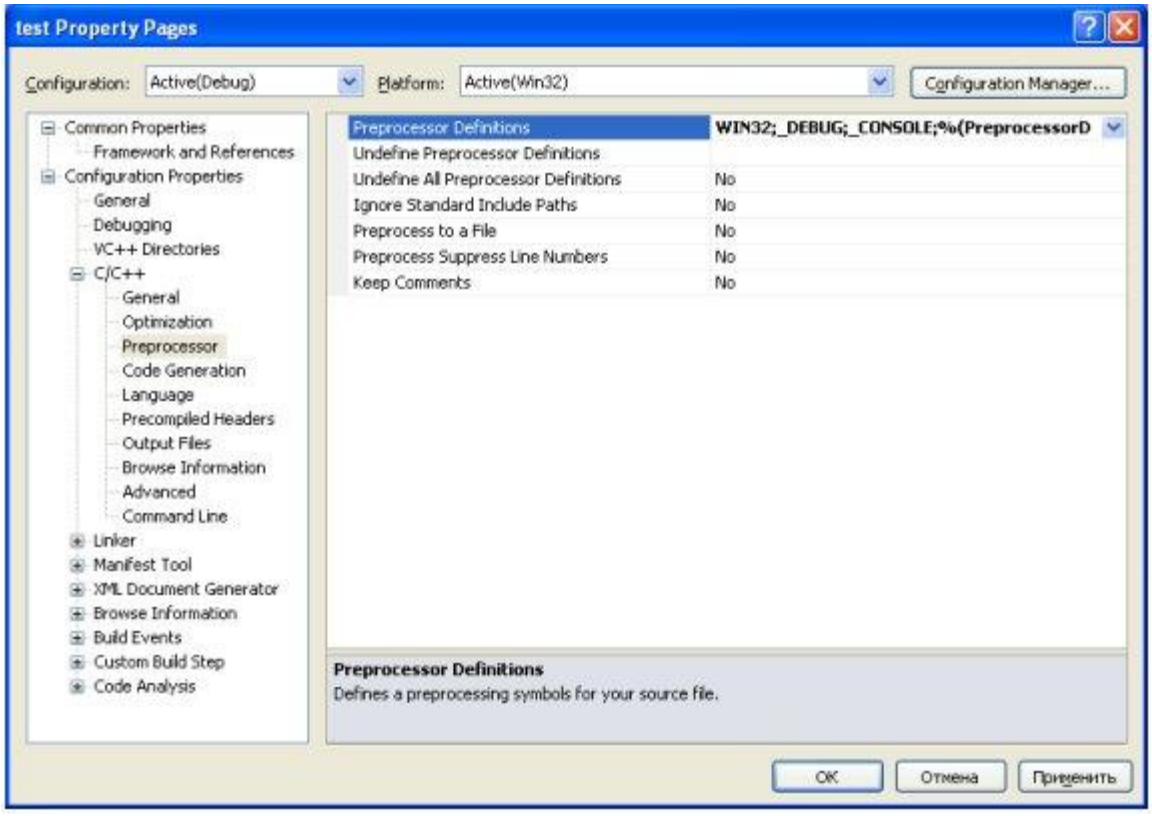

Рис. 24.7. Страница свойств Preprocessor для отладочной конфигурации

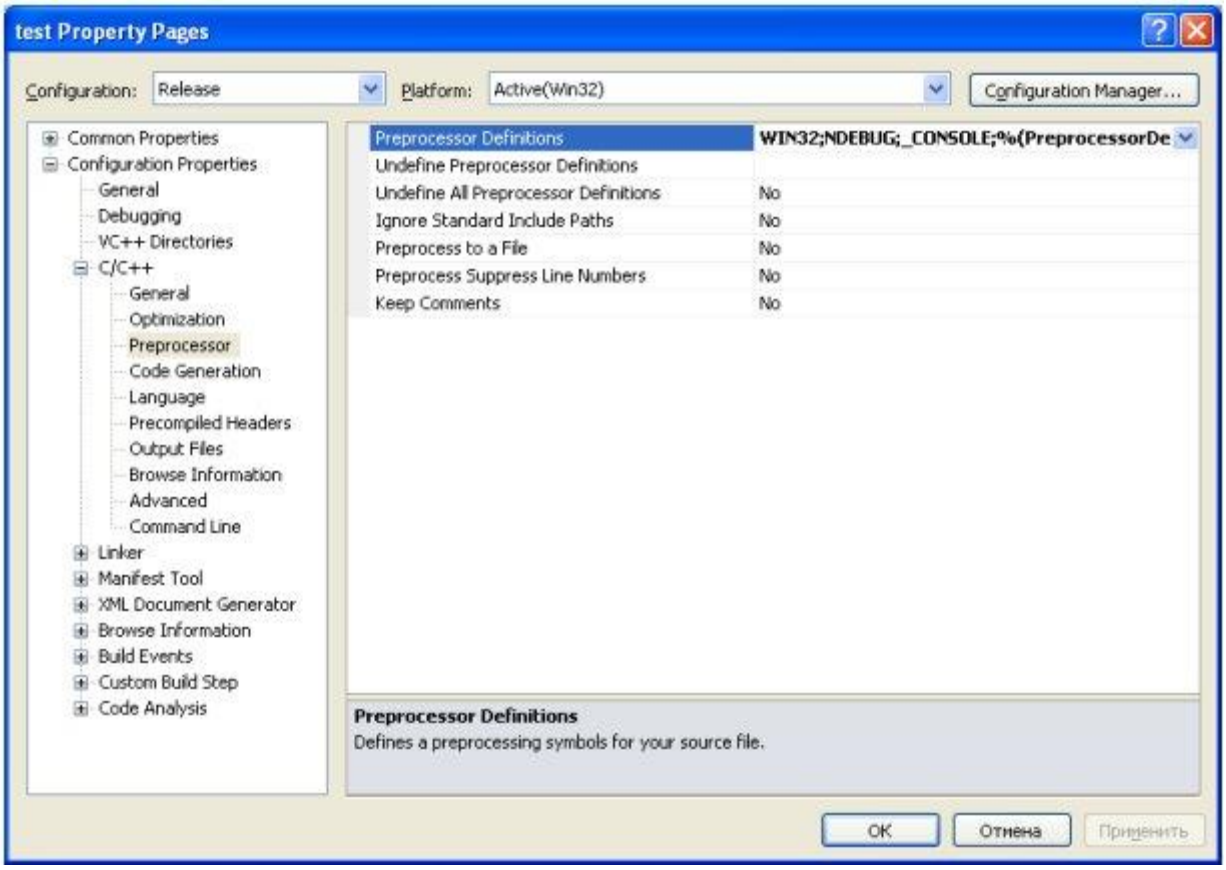

Рис. 24.8. Страница свойств Preprocessor для финальной конфигурации

В отладочной и финальной версиях также различаются форматы отладочной информации (*Debug Information Format*), генерируемой компилятором и сохраняемой в .pdb файле.

Для отладочной версии используется Program *Database* for Edit *and Continue*, позволяющая отлаживать и даже изменять программу, если сработала *точка останова*. При возобновлении выполнения программы, внесенные изменения будут автоматически применены, и выполнение продолжится уже с внесенными изменениями. Эта возможность позволяет сократить время, необходимое на остановку и перекомпиляцию программы при нахождении и исправлении ошибок. В тоже время такая настройка несовместима с настройками оптимизации, поэтому может быть использована только в отладочной версии. На рис. 24.9 показана страница свойств *General* для отладочной конфигурации.

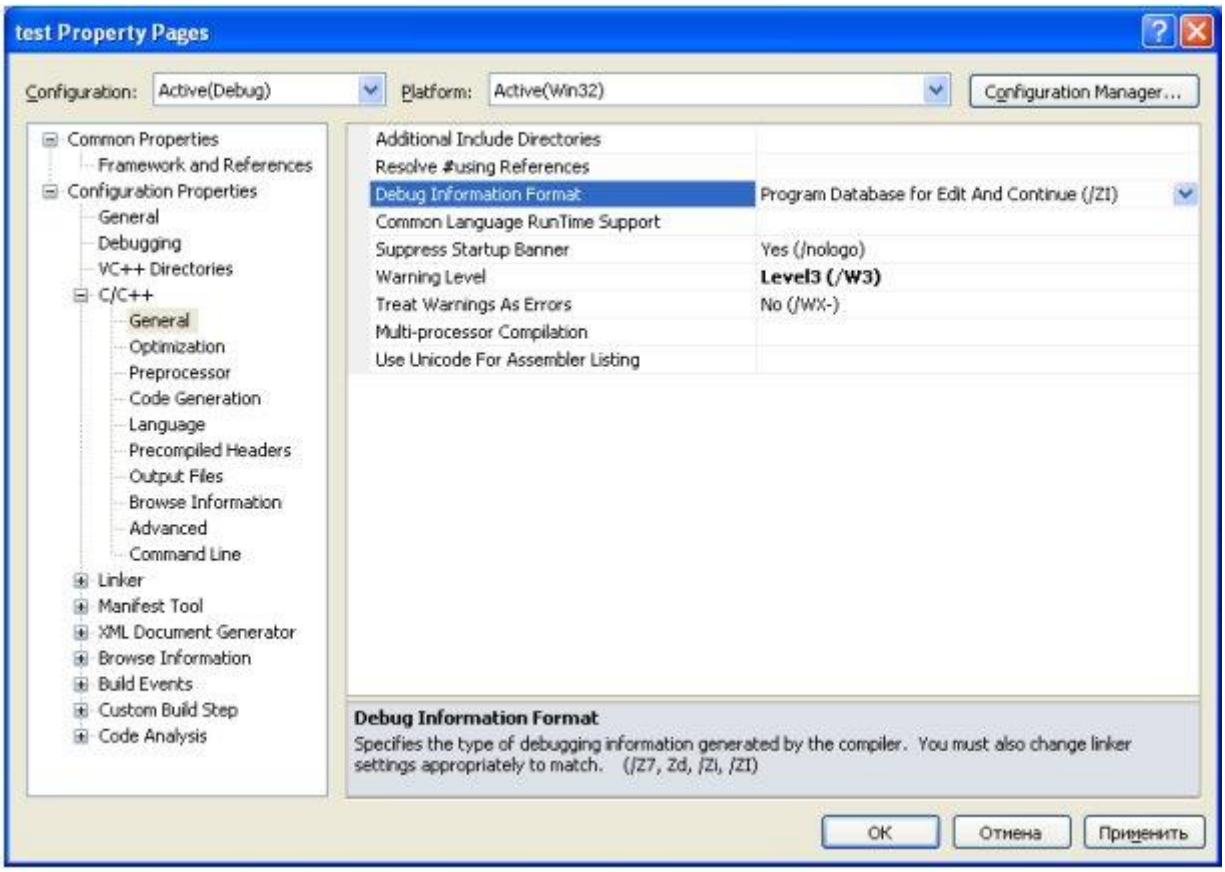

Рис. 24.9. Страница свойств General для отладочной конфигурации

В финальной версии используется настройка Program *Database*. Она включает генерацию .pdb файла, который может быть использован при необходимости поиска ошибок в финальной версии продукта. Эта настройка никак не влияет на оптимизацию генерируемого кода, поэтому она может быть использована для финальной версии.

На рис. 24.10 показана страница свойств *General* для финальной конфигурации.

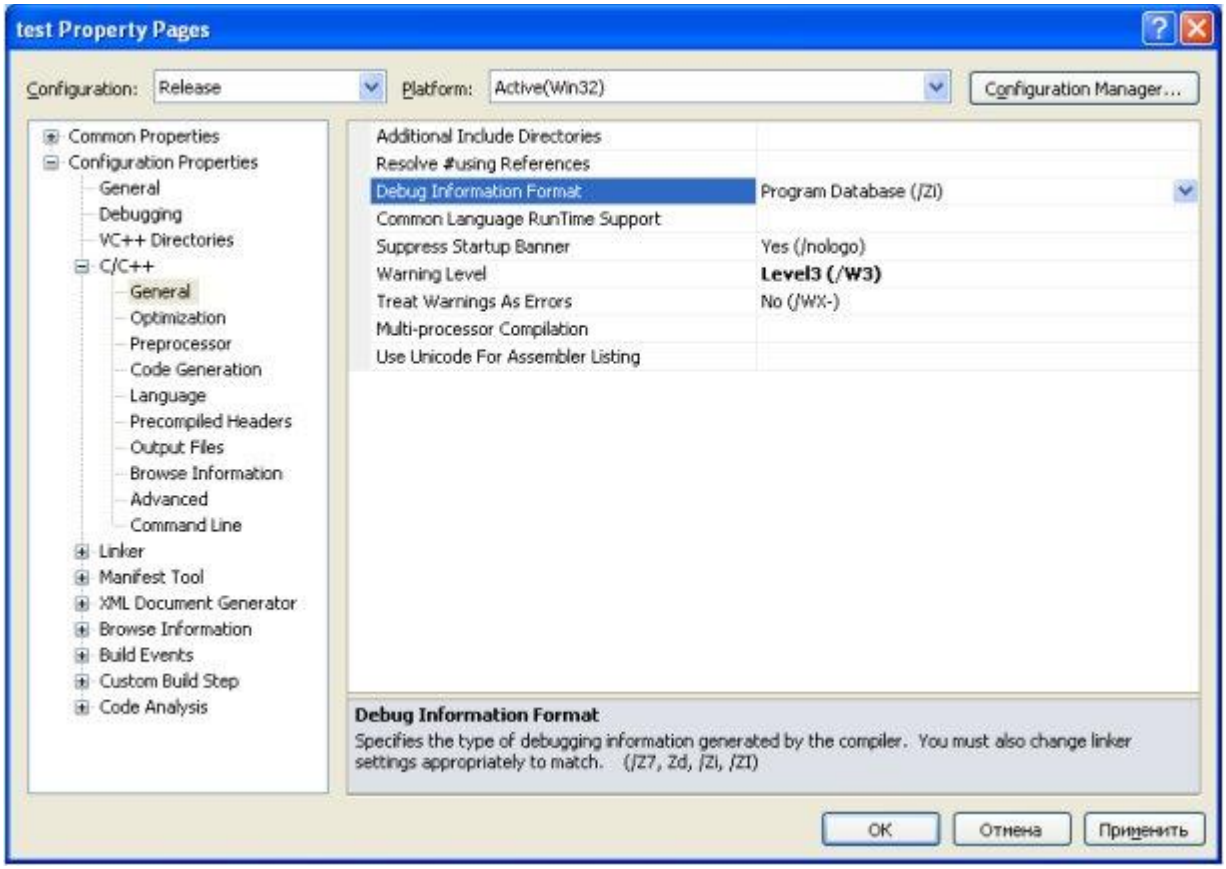

Рис. 24.10. Страница свойств General для финальной конфигурации

На странице свойств *Debugging* (*отладка*) узла Linker настройка *Generate Debug Info* (генерировать отладочную информацию) управляет генерацией отладочной информации. Настройка *Generate* Program *Database* File (создавать *файл* с отладочной информаций для программы) задает имя результирующего .pdb файла с отладочной информацией.

На рис. 24.11 показана страница свойств *Debugging* узла Linker для отладочной версии.

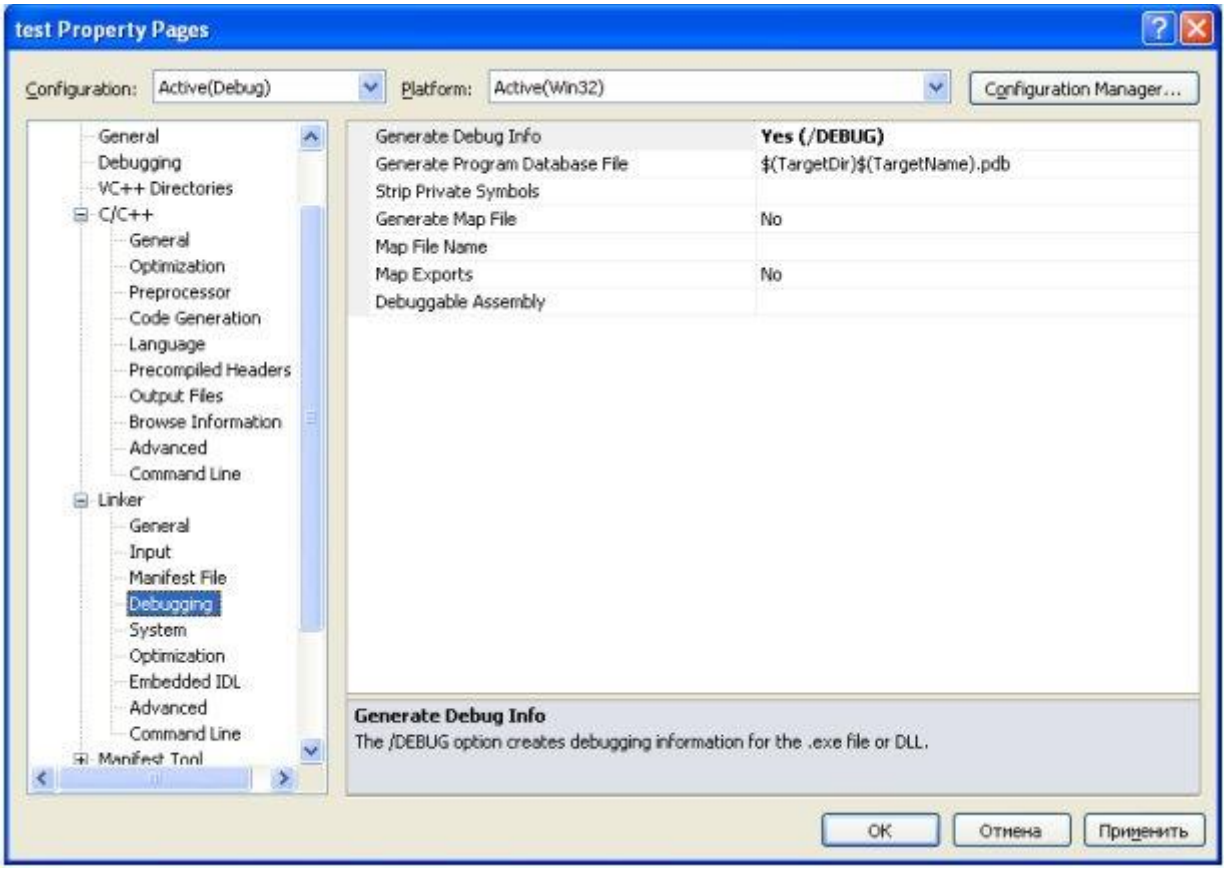

Рис. 24.11. Страница свойств Debugging для отладочной версии

MS *Visual Studio* 2010 предоставляет удобные и гибкие *механизмы* настройки свойств конфигураций проектов, что позволяет программистам выполнять компиляцию и сборку своих проектов с актуальным набором настроек.