

Лабораторная работа №1

Тема: «Изучение среды Visual Studio 2010. Настройка компилятора»

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Язык C в основе своей был создан в 1972 г. как язык для операционной системы UNIX. Его автором считается Денис М. Ритчи (Dennis M. Ritchie).

Некоторое время отсутствовала единая политика по стандартизации языка. В начале 1980-х гг. в Американском национальном институте стандартов (ANSI) был сформирован комитет по стандартизации языка C. В 1989 г. работа комитета по языку C была ратифицирована, и в 1990 г. вышел в свет первый официальный документ по стандарту языка C. Появился стандарт 1989, т. C89. К разработке стандарта по языку C была также привлечена Международная организация по стандартизации (ISO). Появился стандарт ISO/IEC 9899:1990.

Стандарты:

C:

K&R C

C89 | ANSI C | ANSI X3.159-1989

C99 | ISO/IEC 9899:1999

C11 | ISO/IEC 9899:2011

C++:

C++98 | ISO/IEC 14882:1998

<http://www.cplusplus.com/doc/oldtutorial/>

C++03 | ISO/IEC 14882:2003

C++07/TR1 | ISO/IEC TR 19768:2007

C++11 | ISO/IEC 14882:2011

C++14 | ISO/IEC 14882:2014

C++17

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, содержащее 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 писать программу на языке C/C++, необходимо выполнить несколько подготовительных шагов по созданию проекта и выбора и настройки компилятора языка C/C++ для трансляции исходного кода.

После запуска **Microsoft Visual Studio 2010** появляется стартовая страница (рисунок 1).

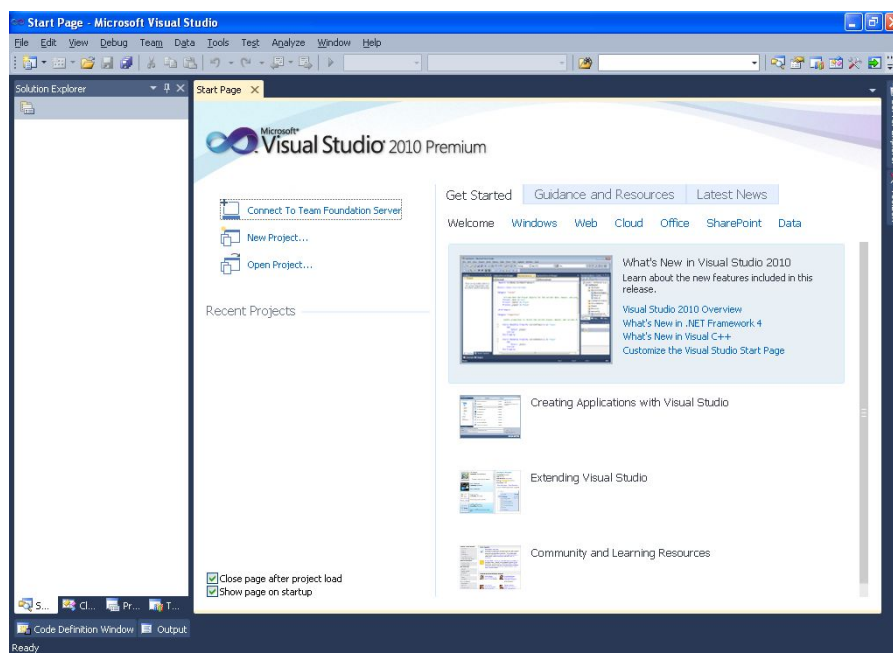


Рисунок 1 – Стартовая страница Visual Studio 2010

Следующим шагом является создание нового проекта. Для этого в меню **File** необходимо выбрать **New Project** (или нажать комбинацию клавиш **Ctrl + Shift + N**). Результат выбора пунктов меню для создания нового проекта показан на рисунке 2.

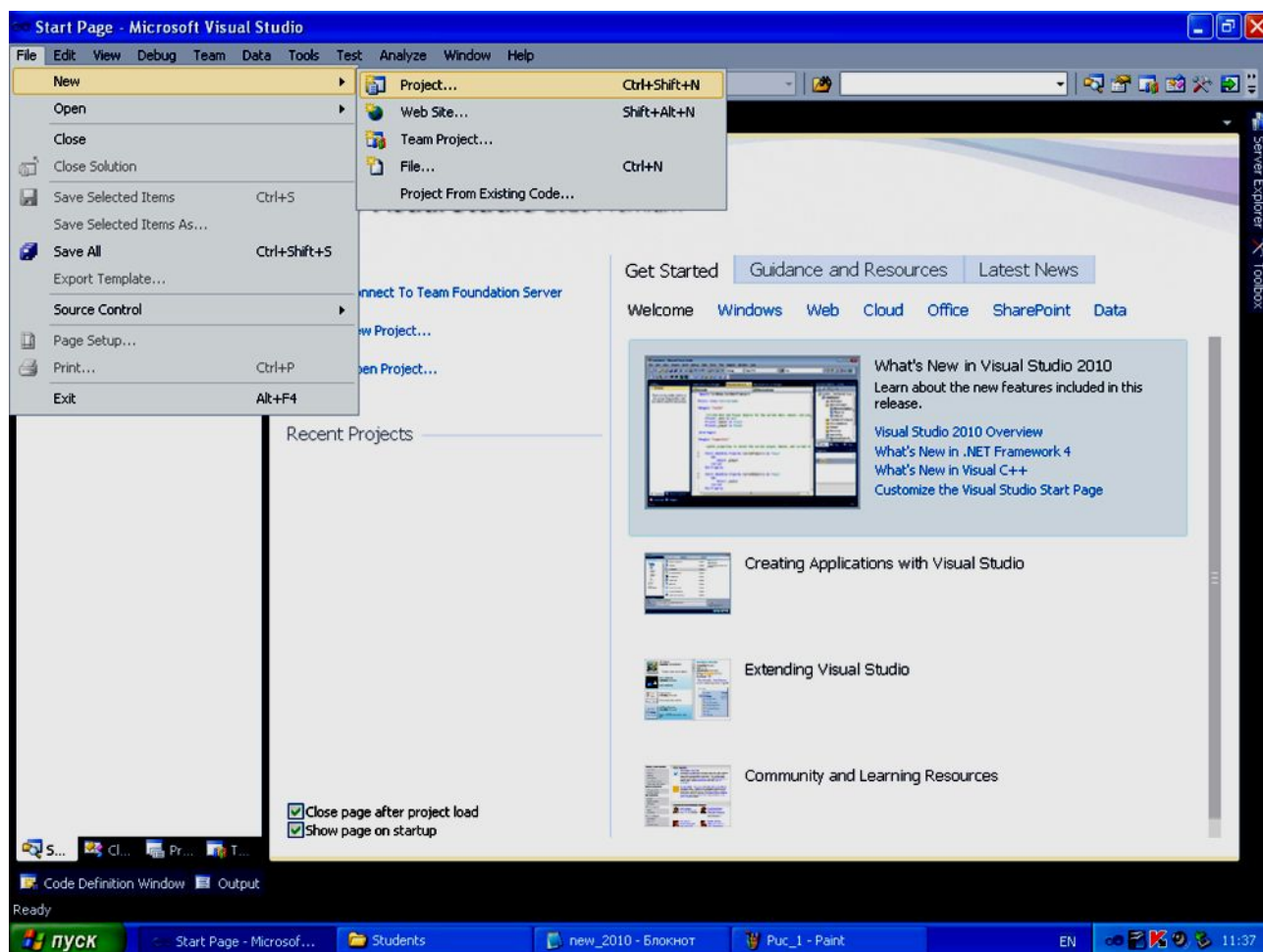


Рисунок 2 – Окно с выбором нового проекта

Среда Visual Studio отобразит окно **New Project** (Создать проект), 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++**, затем обратиться к пункту **Win32** и на центральной панели выбрать **Win32 Console Application** (рисунок 3).

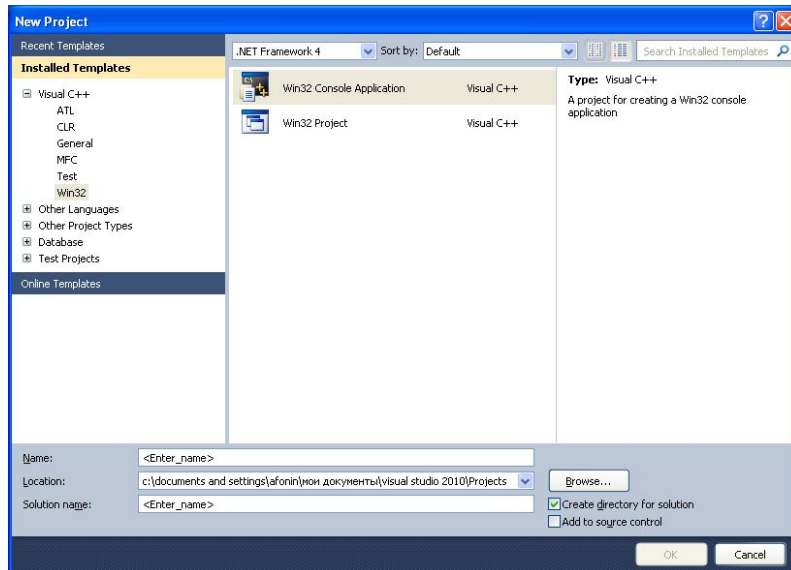


Рисунок 3 – Выбор типа проекта

После выбора типа проекта в поле редактора **Name** необходимо ввести его имя, например **hello**. В поле **Location** можно указать путь размещения проекта или выбрать его (путь) с помощью клавиши (кнопки) **Browse**. По умолчанию проект сохраняется в специальной папке **Projects**. Выбор имени проекта может быть достаточно произвольным: допустимо использовать числовое значение, допустимо имя задавать через буквы русского алфавита. В дальнейшем будем давать проекту имя, набранное с помощью букв латинского алфавита и, может быть, с добавлением цифр.

Пример выбора имени проекта показан на рисунке 4.

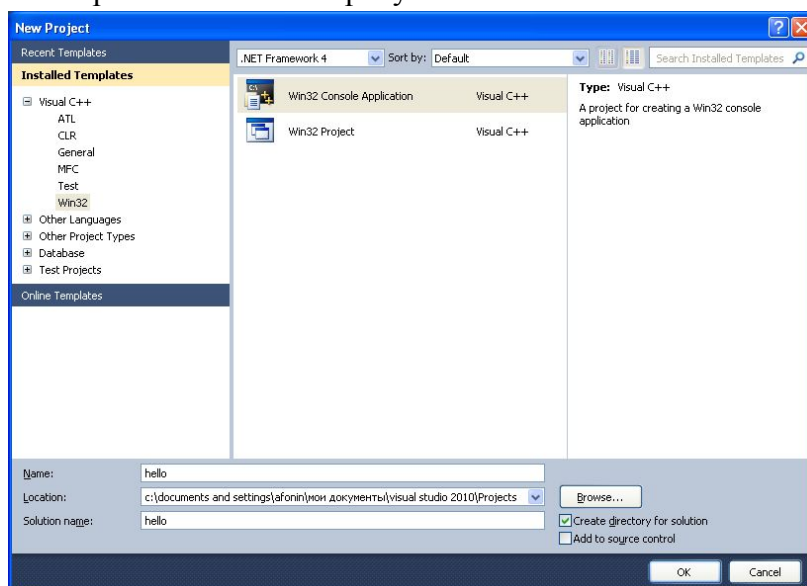


Рисунок 4 – Пример задания имени проекта

Одновременно с проектом Visual Studio создает решение. *Решение* (solution) – это способ объединения нескольких проектов для организации более удобной работы с ними. После нажатия кнопки **ОК** откроется окно **Win32 Application Wizard** (мастер создания приложений для операционных систем Windows) (рисунок 5)

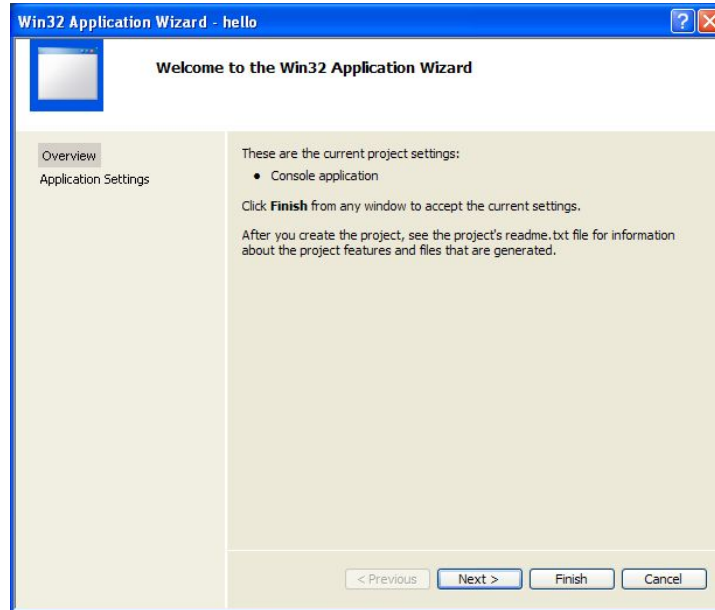


Рисунок 5 – Мастер создания приложения

На первой странице мастера представлена информация о создаваемом проекте, на второй можно сделать его первичные настройки. После обращения к странице **Application Settings** или нажатия кнопки **Next** получим окно, представленное на рисунке 6.

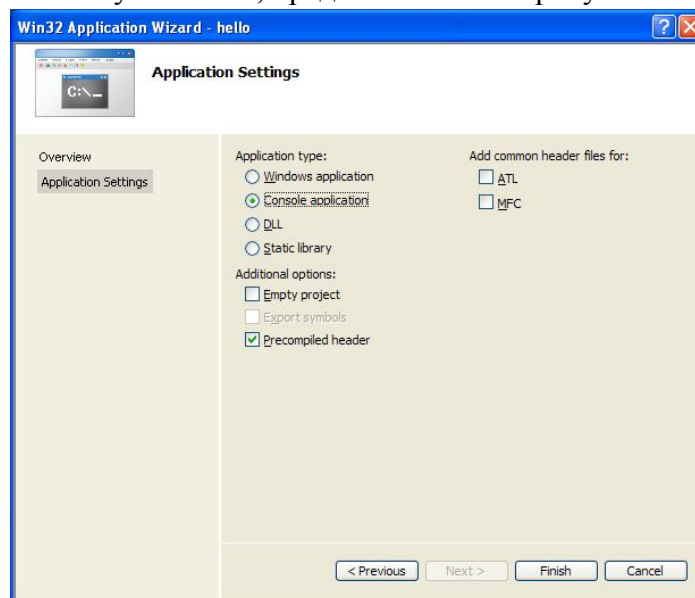


Рисунок 6 – Страница мастера настройки проекта по умолчанию

В дополнительных опциях (**Additional options**) следует поставить галочку в поле **Empty project** (пустой проект) и снять (убрать) ее в поле **Precompiled header** (рисунок 7)

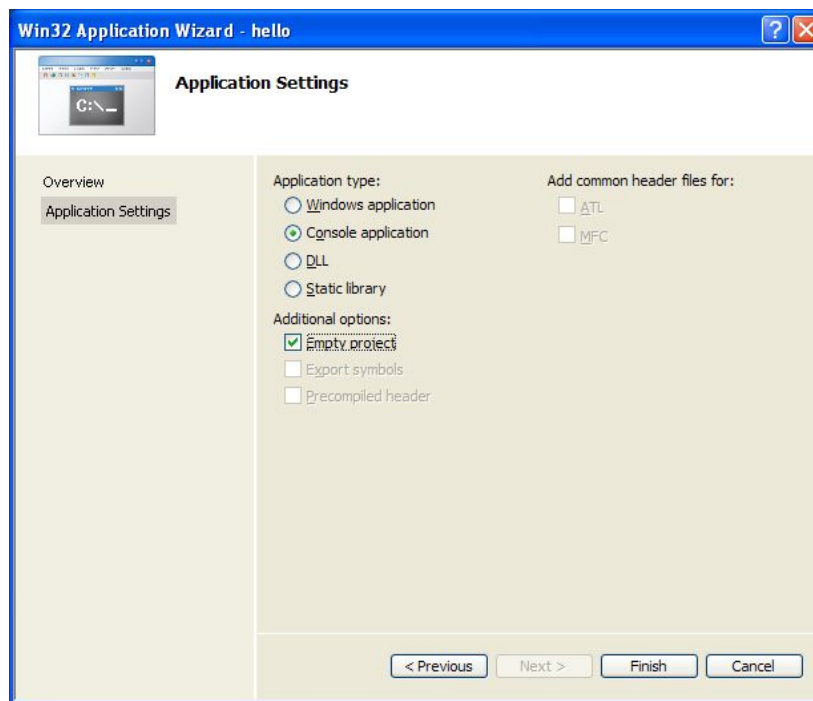


Рисунок 7 – Выполненная настройка мастера приложений

Создадим проект по приведенной схеме, т. е. проект в консольном приложении, который выполняется целиком программистом (за счет выбора **Empty project**). После нажатия кнопки **Finish** получим экранную форму (рисунок 8), где приведена последовательность действий добавления файла для создания исходного кода к проекту. Стандартный путь для этого: подвести курсор мыши к папке **Source Files** (файлы исходного кода) из узла **hello** в левой части открытого проекта приложения, щелкнуть правой кнопкой мыши и выбрать **Add**, затем **New Item** (создать новый элемент).

Экран Visual C++ разделен на четыре основные зоны. Сверху расположены **меню** и **панели инструментов**. Кроме них рабочий стол Visual C++ включает в себя три окна:

Окно **Project Workspace** (окно рабочей области) – расположено в левой части. Первоначально окно закрыто, но после создания нового проекта или загрузки существующего проекта это окно будет содержать несколько вкладок.

Окно **Editor** (окно редактирования), расположено справа. Его используют для ввода, проверки и редактирования исходного кода программы.

Окно **Output** (окно вывода) служит для вывода сообщений о ходе компиляции, сборки и выполнения программы и сообщений о возникающих ошибках.

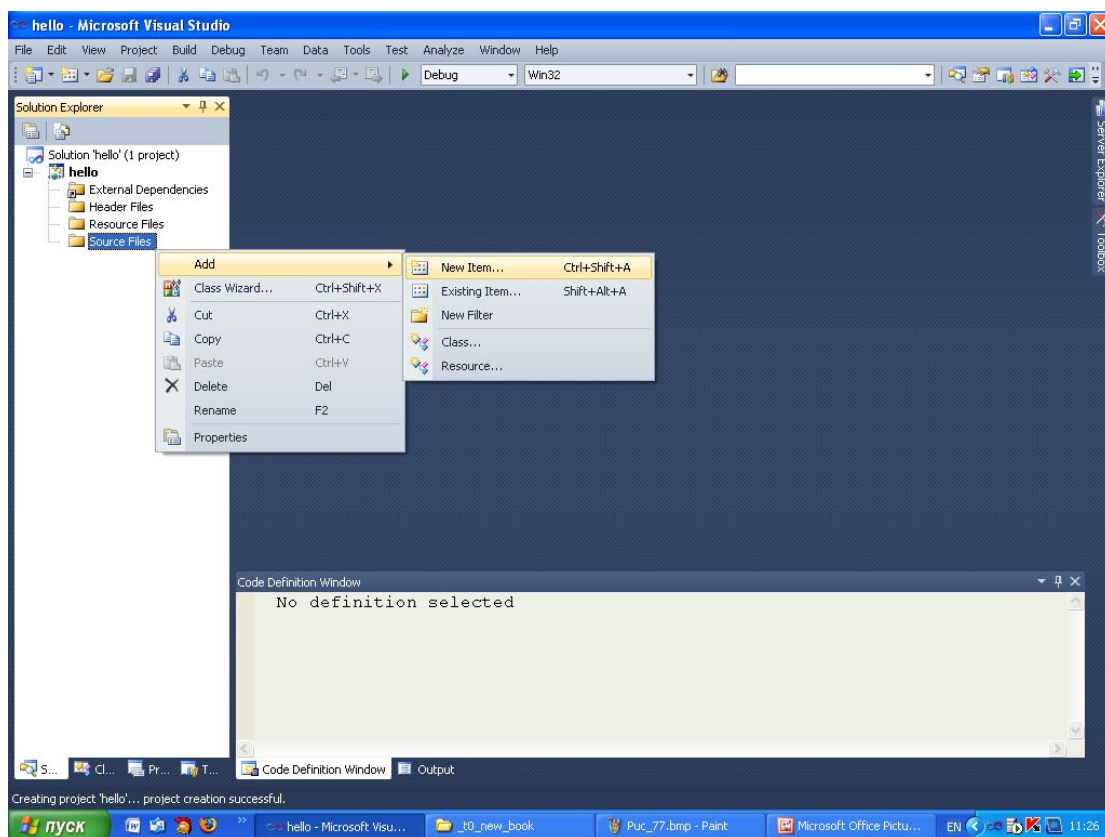
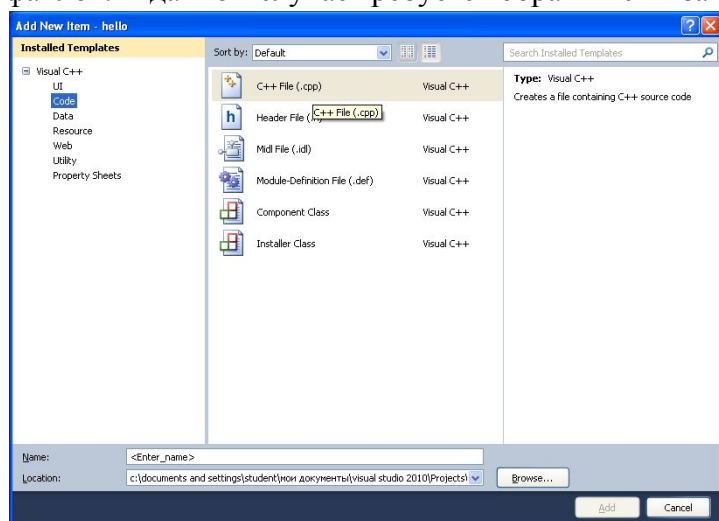


Рисунок 8 – Меню добавления нового элемента к проекту

В результате получим окно (рисунок 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)**.



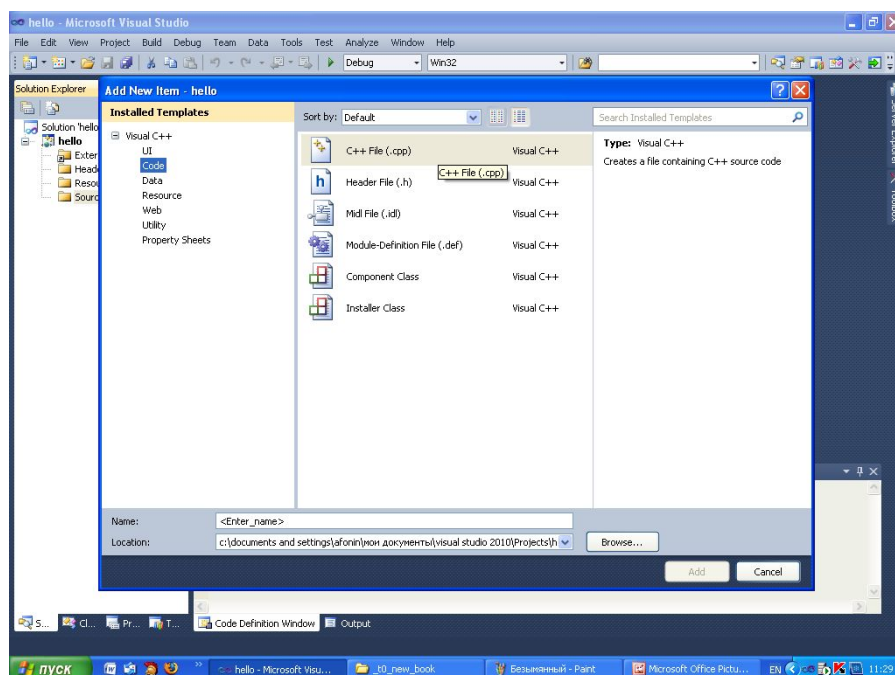


Рисунок 9 – Окно выбора типа файла для подключения к проекту

Теперь в поле редактора **Name** (в нижней части окна) следует задать имя нового файла и указать расширение **.cpp**, например, **main.cpp**. Имя может быть достаточно произвольным, но имеется негласное соглашение, что оно должно отражать назначение файла и логически описывать исходный код, который в нем содержится. В проекте, состоящем из нескольких файлов, есть смысл выделить файл, содержащий главную функцию программы, т. е. ту, с которой она начнет выполняться. Такому файлу будем задавать имя **main.cpp**, где расширение **.cpp** указывает на то, что этот файл содержит исходный код на языке **C++**, и он будет транслироваться соответствующим компилятором. Программам на языке **C++** принято давать указанное расширение. После задания имени файла в поле редактора **Name** получим форму, приведенную на рисунке 10.

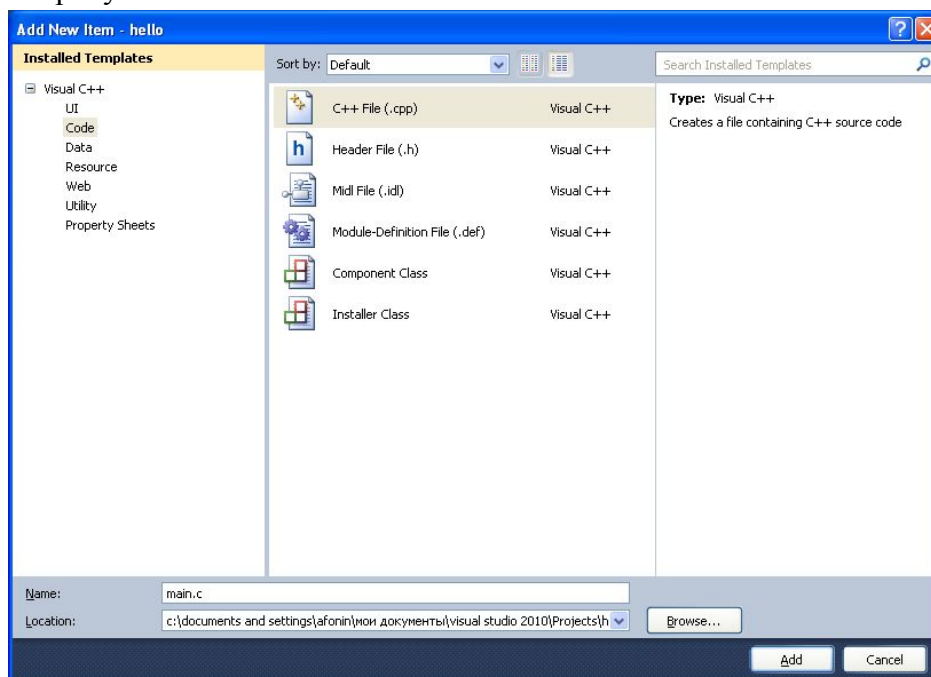


Рисунок 10 – Задание имени файла, подключаемого к проекту

Затем следует нажать кнопку **Add**. Вид среды Visual Studio после добавления первого файла к проекту показан на рисунке 11. Добавленный файл отображается в дереве **Solution Explorer** (обозреватель решений) под узлом **Source Files** (файлы с исходным кодом), и для него автоматически открывается редактор.

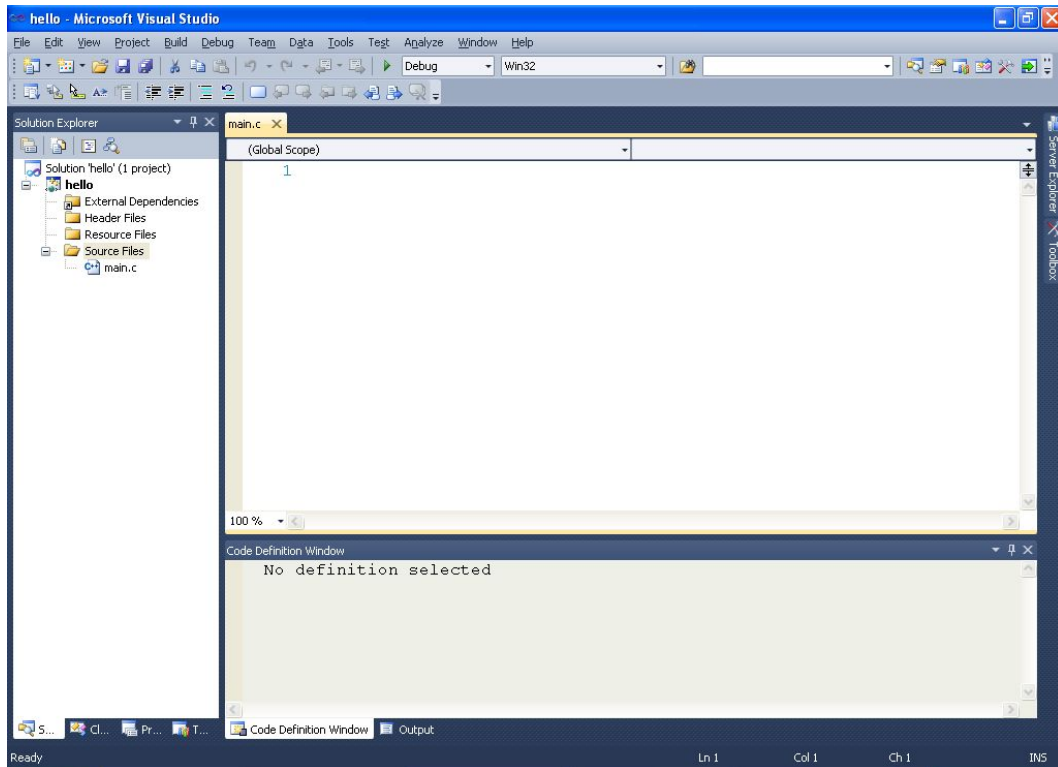


Рисунок 11 – Подключение файла проекта

На рисунке 11 в левой панели в папке **Solution Explorer** отображаются файлы, включенные в проект в папках. Приведем их описание.

Папка **Source Files** предназначена для файлов с исходным кодом. В ней отображаются файлы с расширением **.cpp**.

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

В папке **Resource Files** (файлы ресурсов) представлены файлы ресурсов, например изображения и т. д.

Папка **External Dependencies** (внешние зависимости) отображает файлы, не добавленные явно в проект, но используемые в файлах исходного кода, например включенные при помощи директивы **#include**. Обычно в этой папке присутствуют заголовочные файлы стандартной библиотеки, применяющиеся в проекте.

Следующий шаг состоит в настройке проекта. Для этого в меню **Project** главного меню следует выбрать **hello Properties** (или одновременно нажать клавиши **Alt + F7**). Пример обращения к этому пункту меню показан на рисунке 12.

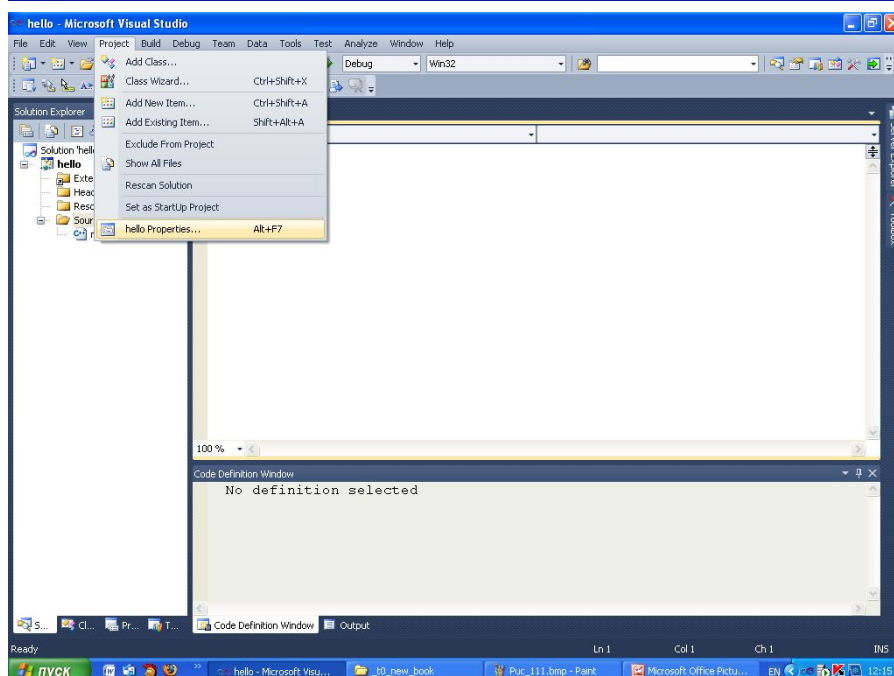
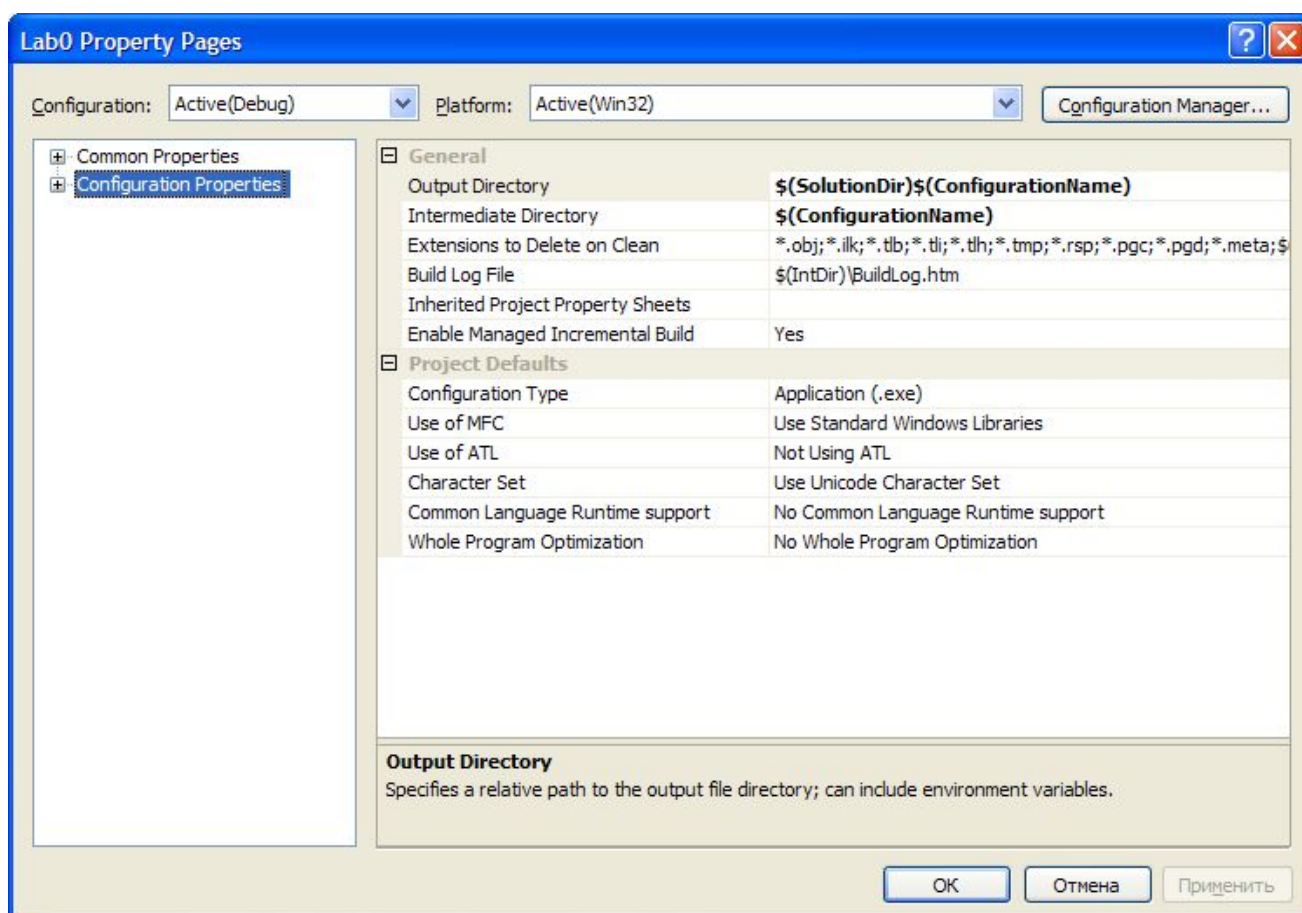


Рисунок 12 – Обращение к странице свойств проекта

После того как откроется окно свойств проекта, следует обратиться (с левой стороны) к **Configuration Properties** (Свойства конфигурации) ; появится ниспадающий список (рисунок 13) Далее нужно обратиться к узлу **Project Defaults** (значения по умолчанию для проекта) и через него в левой панели выбрать **Character Set** (набор символов), где установить свойство **Use Multi-Byte Character Set**. Настройка **Character Set** (набор символов) позволяет определиться, какая кодировка символов – ANSI или UNICODE – будет использована при

компиляции программы. Для совместимости со стандартом C89 можно выбрать **Use Multi-Byte Character Set**. Это позволяет применять многие привычные функции, например по выводу информации на консоль.

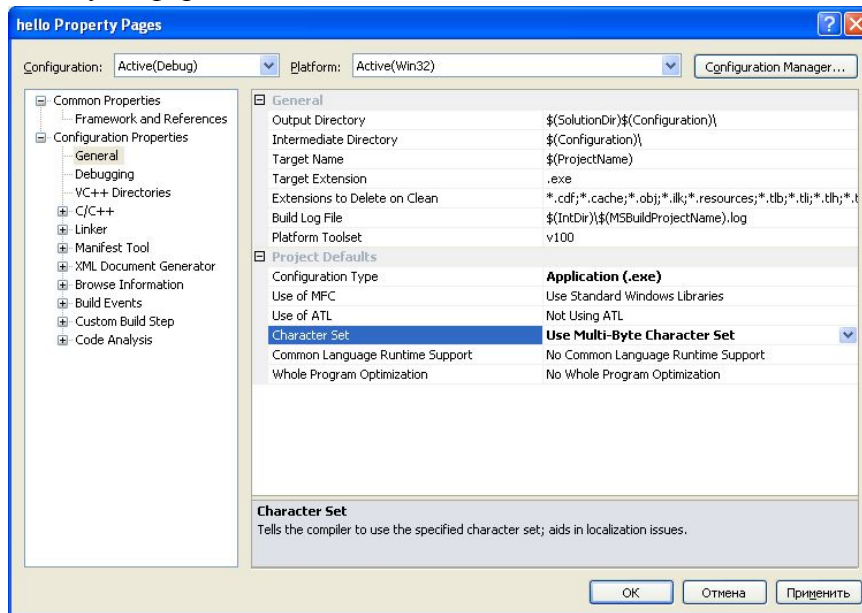


Рисунок 13 – Меню списка свойств проекта

После осуществления выбора (см. рисунок 13) следует нажать кнопку **Применить**.

Затем необходимо выбрать узел **C/C++** и в ниспадающем меню выбрать пункт **Code Generation** (создание кода), через который в правой части панели обратиться к закладке **Enable C++ Exceptions** (включить C++ исключения), установив для нее **No** (запрещение исключений C++). Результат установки выбранного свойства представлен на рис. 14. Затем нужно нажать кнопку **Применить**.

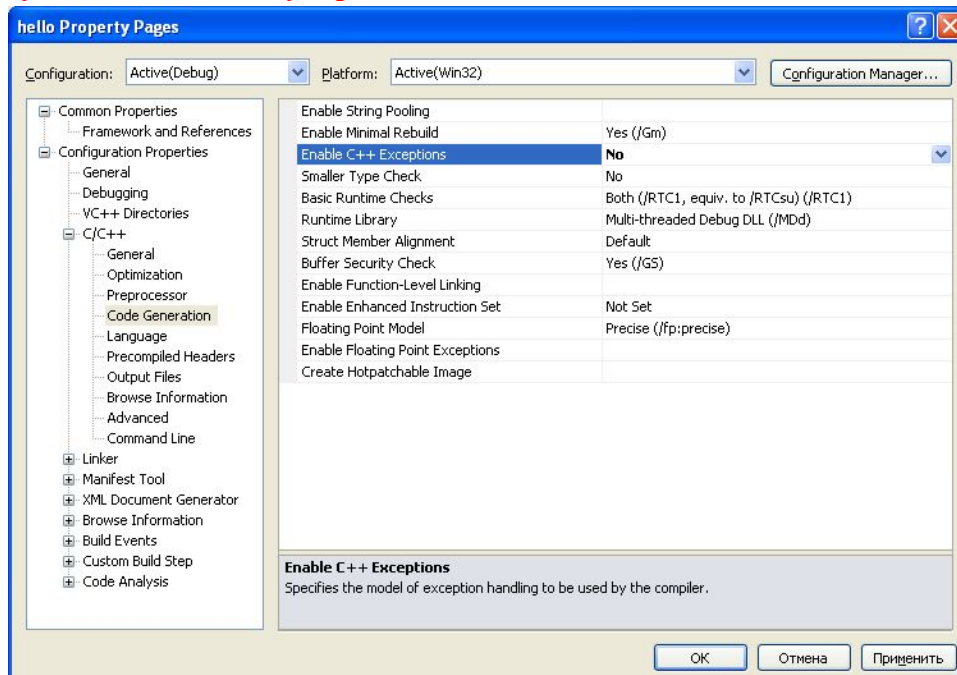


Рисунок 14 – Страница свойств для запрещения исключений C++

Далее в ниспадающем меню узла **C/C++** необходимо выбрать пункт **Language** и через него обратиться в правую часть панели, где установить следующие значения:

для свойства **Disable Language Extensions** (отключить расширения языка) – **Yes (/Za)**, для **Treat Wchar_t As Built in Type** (считать тип wchar_t как встроенный тип) – **No (/Zc:wchar_t-)**, для **Force Conformance in For Loop Scope** (соответствие стандарту определения локальных переменных в операторе цикла for) – **Yes (Zc:forScope)**, для **Enable Run-Time Type Information** (разрешить информацию о типах во время выполнения) – **No (/GR-)**, для свойства **Open MP Support** (разрешить расширение Open MP; используется при написании программ для многопроцессорных систем) – **No (/openmp-)**. Результат выполнения этих действий показан на рисунке 15.

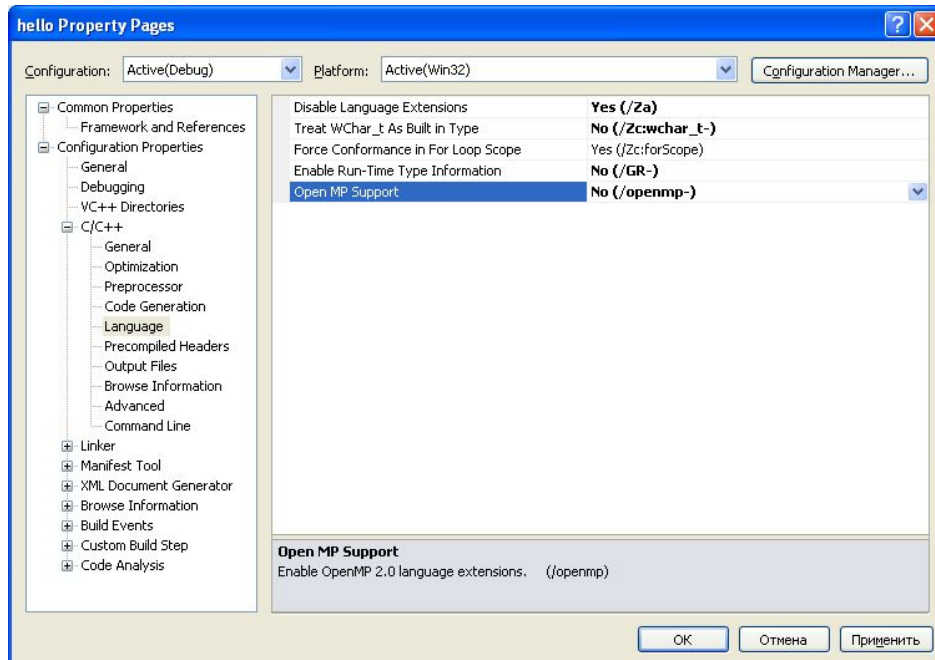


Рисунок 15 – Страница свойств закладки Language

После выполнения указанных действий следует нажать клавишу **Применить**. Далее в ниспадающем списке узла C/C++ следует выбрать пункт **Advanced** (дополнительно) и в правой панели изменить свойство **Compile As** (компилировать как) в свойство компиляции языка C, т.е. **Compile as C Code (/TC)**, если вы работаете с программой на языке C. Или в ниспадающем списке узла C/C++ следует выбрать пункт **Advanced** (дополнительно) и в правой панели изменить свойство **Compile As** (компилировать как) в свойство компиляции языка, т.е. **Compile as C++ Code (/TP)**, если вы работаете с программой на языке C++. Результат установки компилятора языка C представлен на рисунке 16.1. Результат установки компилятора языка C++ представлен на рисунке 16.2

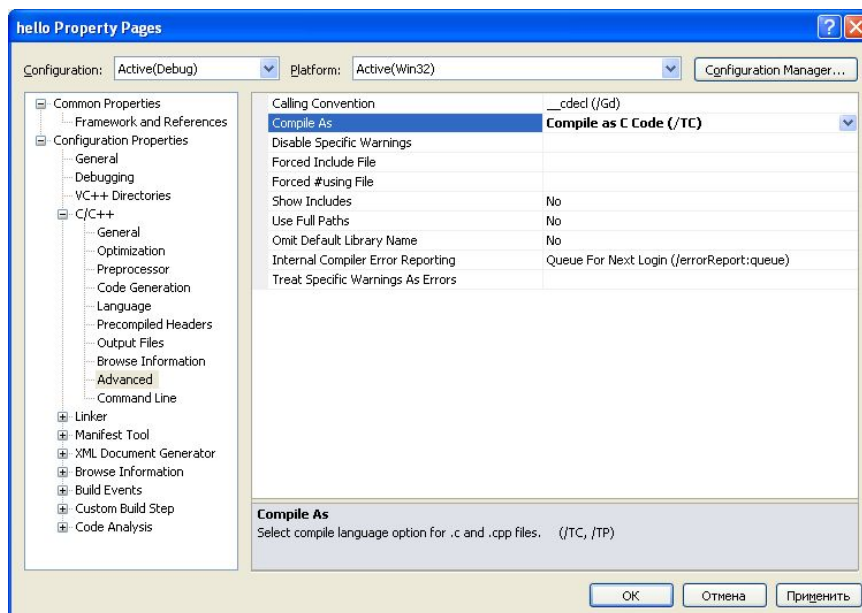


Рисунок 16.1 – Результат выбора режима компиляции языка C

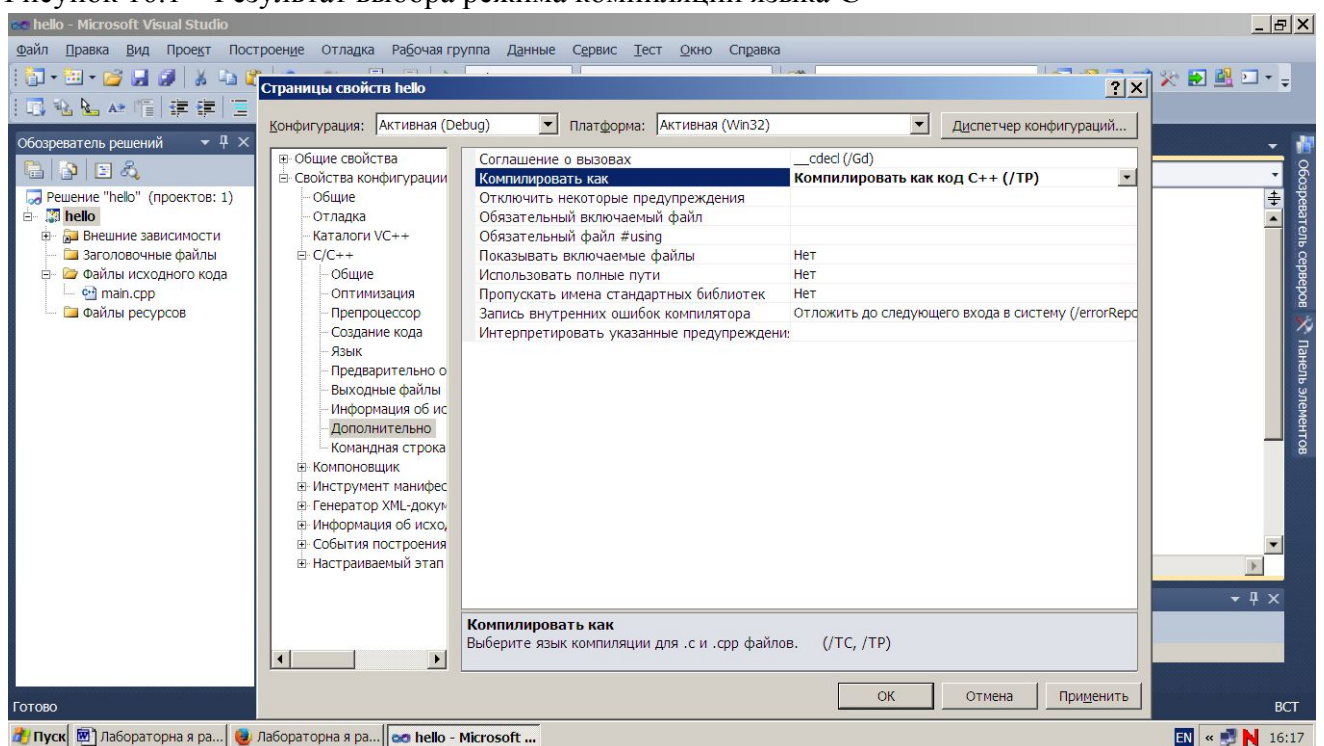


Рисунок 16.2 – Результат выбора режима компиляции языка C++

После нажатия клавиш **Применить** и **ОК** 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 клавиши **Ctrl + F7**. В случае успешной компиляции получим экранную форму, показанную на рисунок 17.1 (пример для кода C). Рисунок 17.2 пример для кода C++.

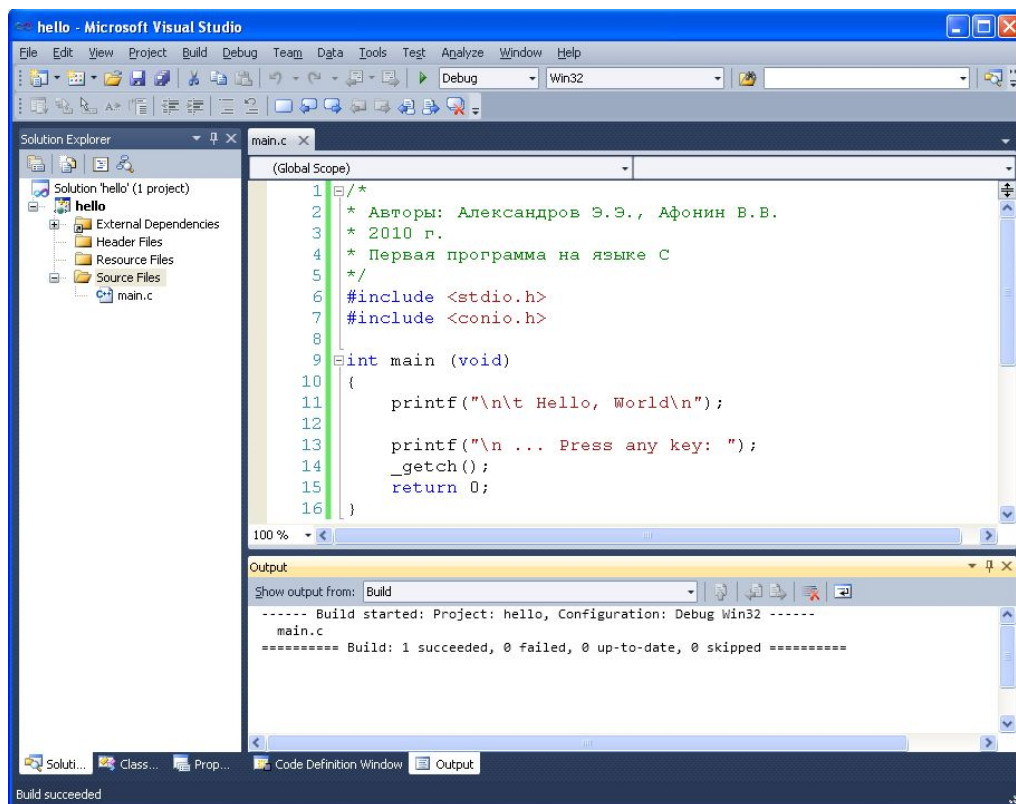


Рисунок 17.1 – Успешно скомпилированная первая программа на языке C

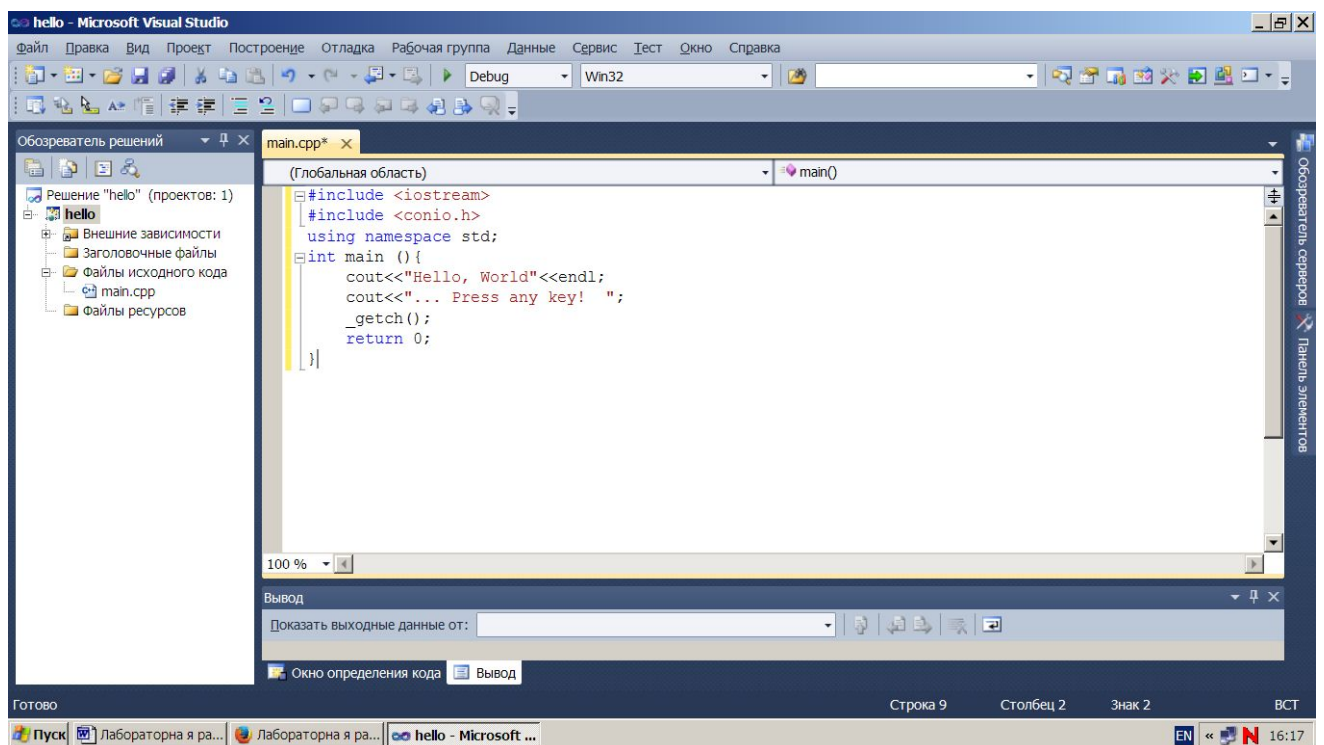


Рисунок 17.1 – Успешно скомпилированная первая программа на языке C

Для приведенного кода программы запуск на ее исполнение из окна редактора в Visual Studio 2010 выполняется нажатием клавиши **F5** (рисунок 18).

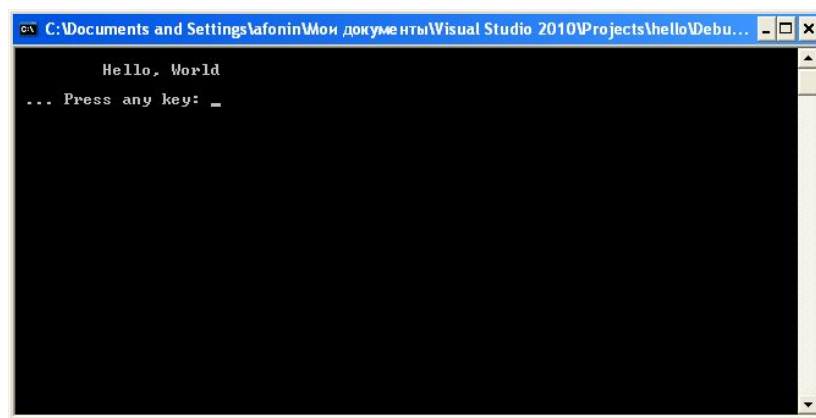


Рисунок 18 – Консольный вывод первой программы на языке C

Произведем разбор первой программы на языке C.

Во-первых, отметим, что в языке C нет стандартных инструкций (операторов) для вывода сообщений на консоль (окно пользователя), а предусмотрены 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 в программу включен также заголовочный файл **conio.h**, поддерживающий функцию **_getch()**. 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 этих скобок отсутствует или в них представлено 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 них, будут относиться к этой функции. В теле функции **main()** имеются еще три инструкции. Вызов функции **printf()**, которая находится в библиотеке компилятора языка C и печатает или отображает те аргументы, которые были подставлены вместо параметров. Символ **\n** означает единый символ newline (новая строка), т.е. с его помощью выполняется 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. предназначена для приема сообщения о нажатии какой-либо (не функциональной) клавиши на клавиатуре. С другими компиляторами, может потребоваться **getch()** 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.

Произведем разбор первой программы на языке C++.

iostream — заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++. Он включён в стандартную библиотеку C++. Название образовано от Input/Output Stream («поток ввода-вывода»). В языке C++ и его

предшественнике, языке программирования Си, нет встроенной поддержки ввода-вывода, вместо этого используется библиотека функций. `iostream` управляет вводом-выводом, как и `stdio.h` в Си. `iostream` использует объекты `cin`, `cout`, `cerr` и `clog` для передачи информации и из стандартных потоков ввода, вывода, ошибок (без буферизации) и ошибок (с буферизацией) соответственно. Являясь частью стандартной библиотеки C++, эти объекты также являются частью стандартного пространства имён — `std`.

using namespace std;

Все типы и функции стандартной библиотеки C++ объявлены в пространстве имен `std` или в пространстве имен, вложенном в `std`.

Пространство имён (англ. `namespace`) — некоторое множество, под которым подразумевается модель, абстрактное хранилище или окружение, созданное для логической группировки уникальных идентификаторов (то есть имён).

Идентификатор, определённый в пространстве имён, ассоциируется с этим пространством. Один и тот же идентификатор может быть независимо определён в нескольких пространствах. Таким образом, значение, связанное с идентификатором, определённым в одном пространстве имён, может иметь (или не иметь) такое же значение, как и такой же идентификатор, определённый в другом пространстве. Языки с поддержкой пространств имён определяют правила, указывающие, к какому пространству имён принадлежит идентификатор (то есть его определение).

Для работы с консолью в программу включен также заголовочный файл **conio.h**, поддерживающий функцию **_getch()**. Строка программы

int main ()

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 этих скобок отсутствует или в них представлено 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 них, будут относиться к этой функции. В теле функции **main()** имеются еще три инструкции. Строка программы

```
cout<<"Hello, World"<<endl;  
cout<<"... Press any key!  ";
```

выводит сообщение на экран.

Библиотека `iostream` определяет три стандартных потока:

- **cin** стандартный входной поток (`stdin` в C)
- **cout** стандартный выходной поток (`stdout` в C)
- **cerr** стандартный поток вывода сообщений об ошибках (`stderr` в C)

Для их использования в Microsoft Visual Studio необходимо прописать строку:

using namespace std;

Для выполнения операций ввода-вывода переопределены две операции поразрядного сдвига:

- `>>` получить из входного потока
- `<<` поместить в выходной поток

Функция без параметров **_getch()** извлекает символ из потока ввода, т.е. предназначена для приема сообщения о нажатии какой-либо (не функциональной) клавиши на клавиатуре. С другими компиляторами, может потребоваться **getch()** 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.

Все файлы проекта сохраняются в той папке, которая сформировалась после указания в поле **Location** имени проекта (hello). На рисунке 19.1 и 19.2 показаны папки и файлы проекта **Visual Studio 2010**.

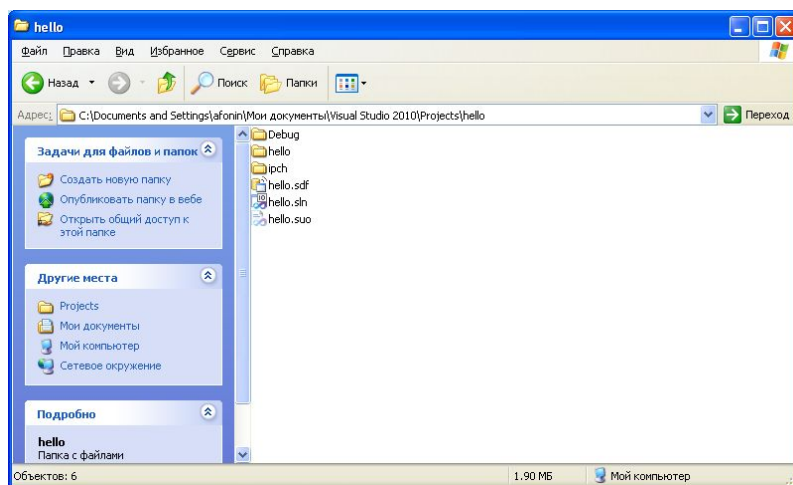
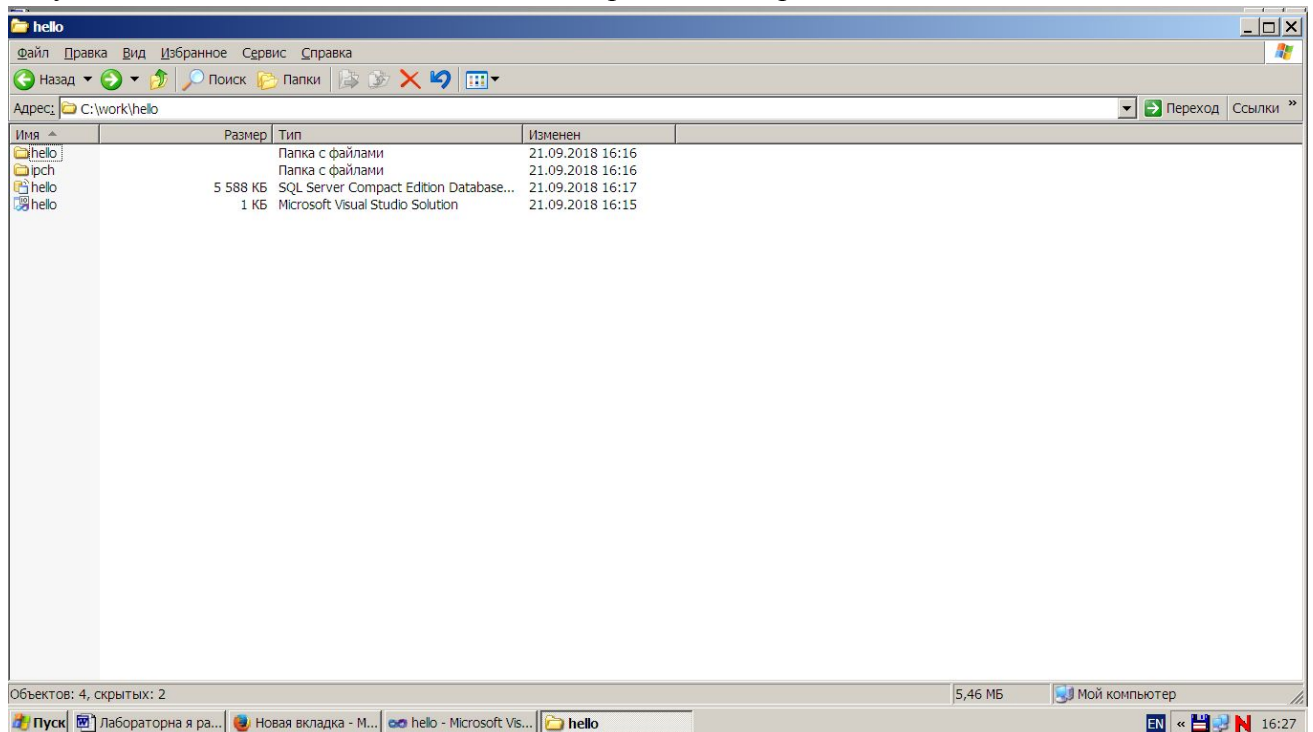


Рисунок 19.1 – Файлы и папки созданного проекта для проекта C



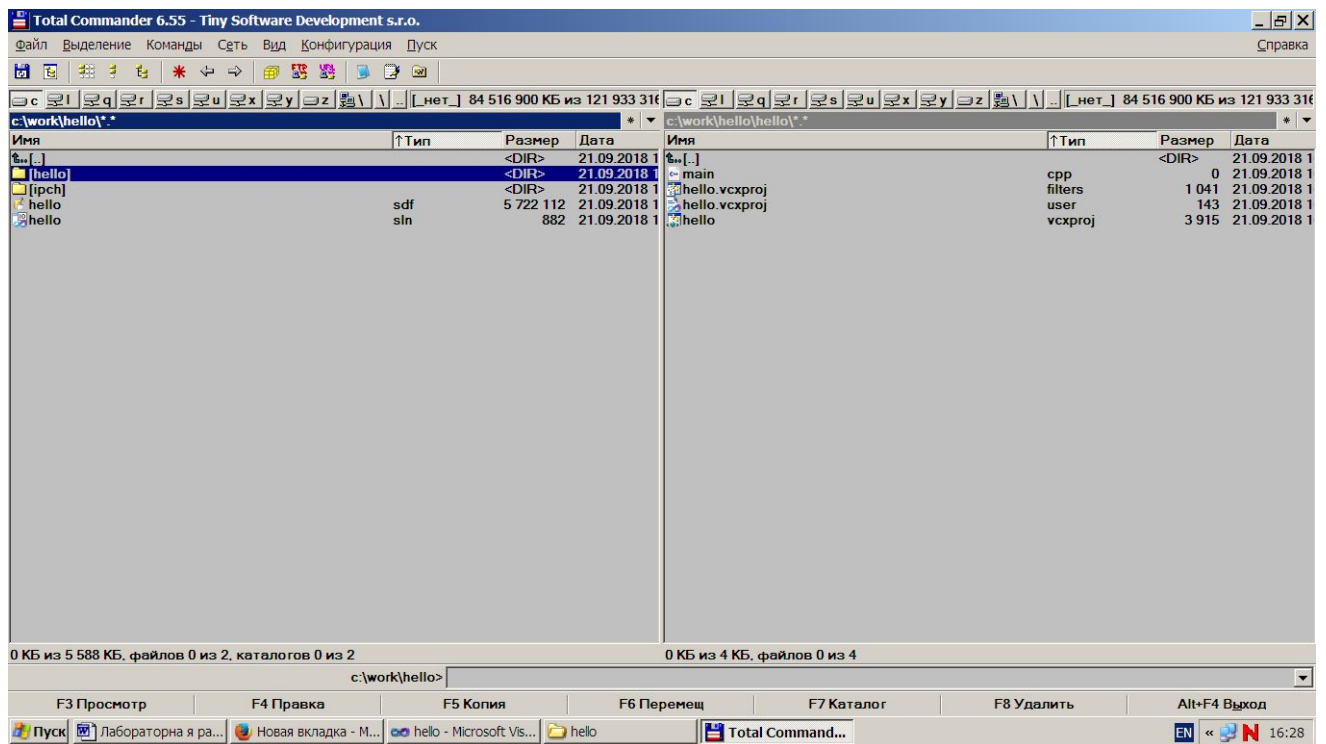


Рисунок 19.1 – Файлы и папки созданного проекта для проекта C++

Каждый файл обладает некоторым значением:

- **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** представлены на рисунке 20.1 и рисунке 20.2

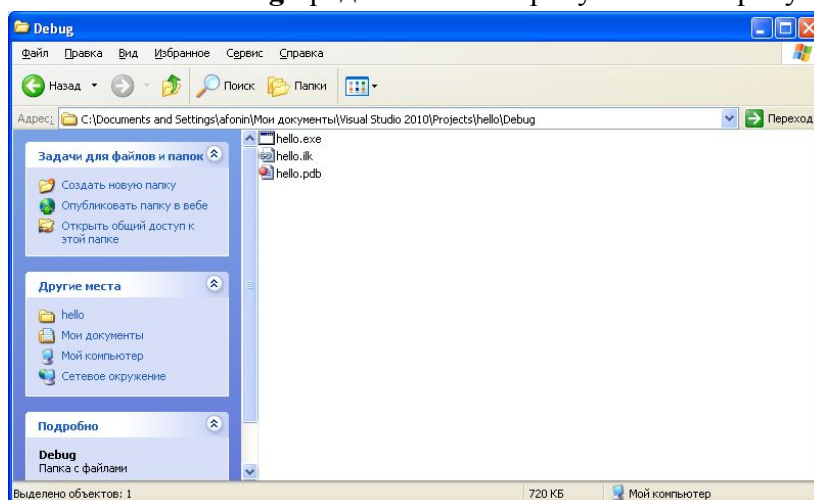


Рисунок 20.1 – Файлы папки Debug для проекта C

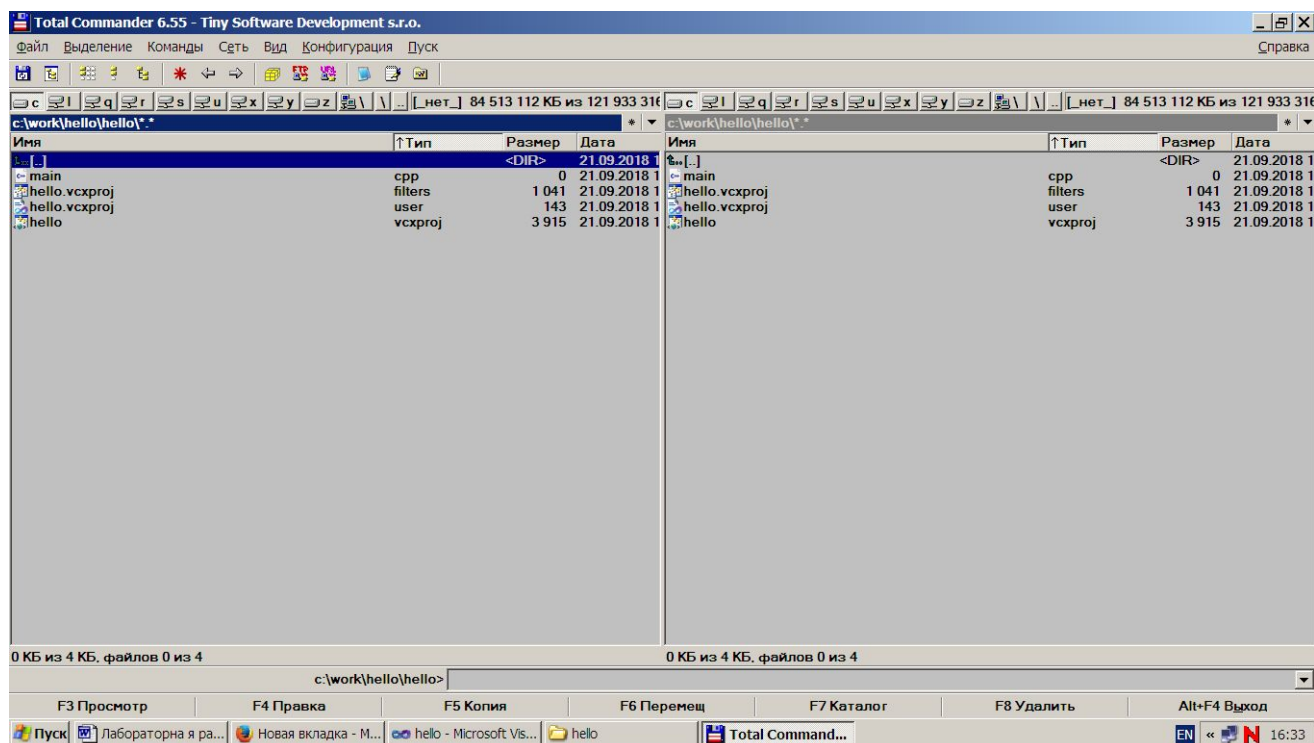


Рисунок 20.2 – Файлы папки Debug для проекта C++

Характеристика содержимого папки **hello**:

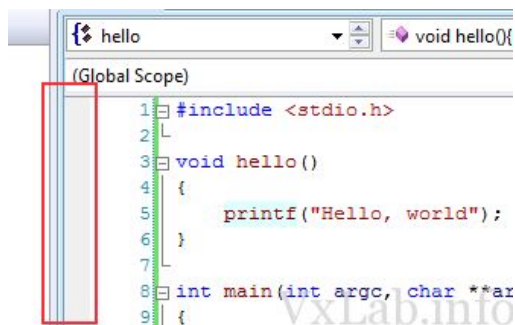
1. **main.c / main.cpp** – файл исходного программного кода;
2. **hello.vcxproj** – файл проекта;
3. **hello.vcxproj.user** – файл пользовательских настроек, связанных с проектом;
4. **hello.vcxproj.filters** – файл с описанием фильтров, используемых Visual Studio Solution Explorer для организации и отображения файлов с исходным кодом.

Отладка в Visual Studio

Отладка – неотъемлемый этап цикла разработки приложений, зачастую более важный, чем написание кода. Именно отладка позволяет устранить проблемные места в коде, которые приводят к разного рода ошибкам.

Точки останова

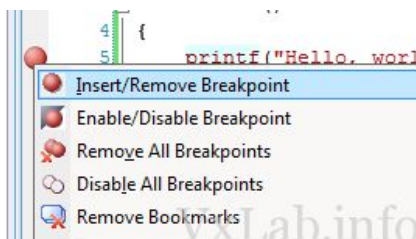
Любой алгоритм выполняется последовательно, одна инструкция за другой. Точка остановки позволяет приостановить выполнение кода на определенной инструкции. Это необходимо, чтобы начать отладку с предположительно проблемного участка. Например, при модульной структуре проекта проблемы начались при подключении нового модуля. Незачем отлаживать весь проект, когда можно отладить только модуль. Точки останова сильно облегчают процесс в этом случае. Для того, чтобы установить точку останова необходимо кликнуть на небольшую область, выделенную как правило особым цветом, в левой части редактора кода. Также это можно сделать, установив каретку на нужную строку и нажав F9.



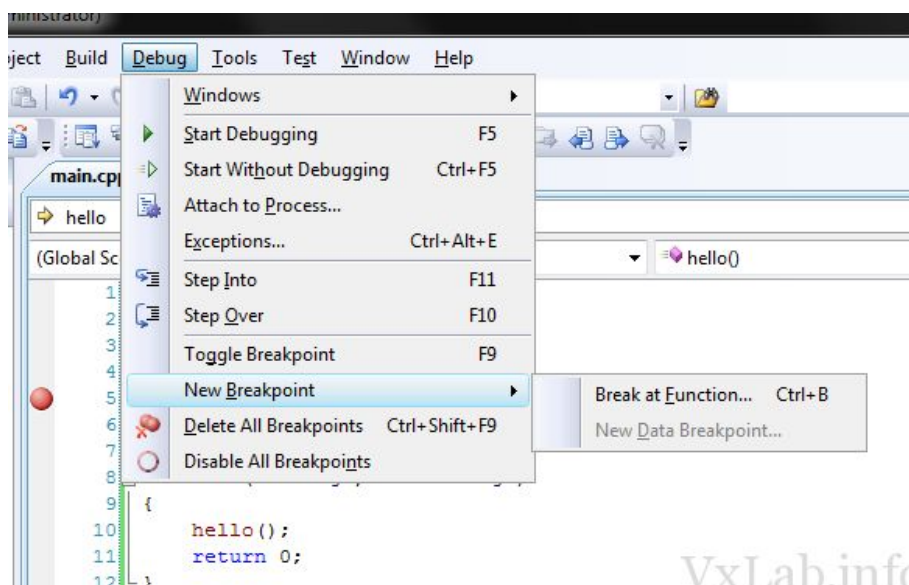
При установке точки останова в окне редактора кода будет выведена иконка напротив строки кода.



Клик левой кнопкой мыши по иконке приведет к удалению точки остановки, а для управления необходимо открыть контекстное меню (клик правой кнопкой).

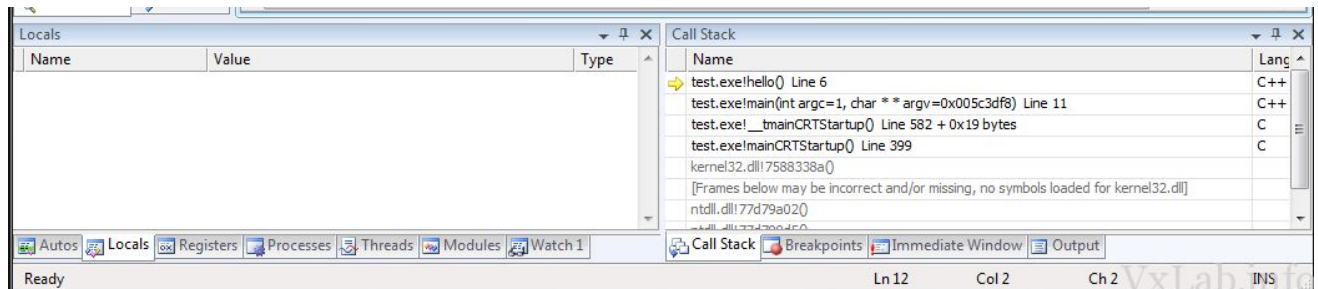


Среди доступных действий также можно отключить и удалить все точки остановки. Те же действия возможно выполнить из вкладки **Debug**, либо по нажатию соответствующих горячих клавиш

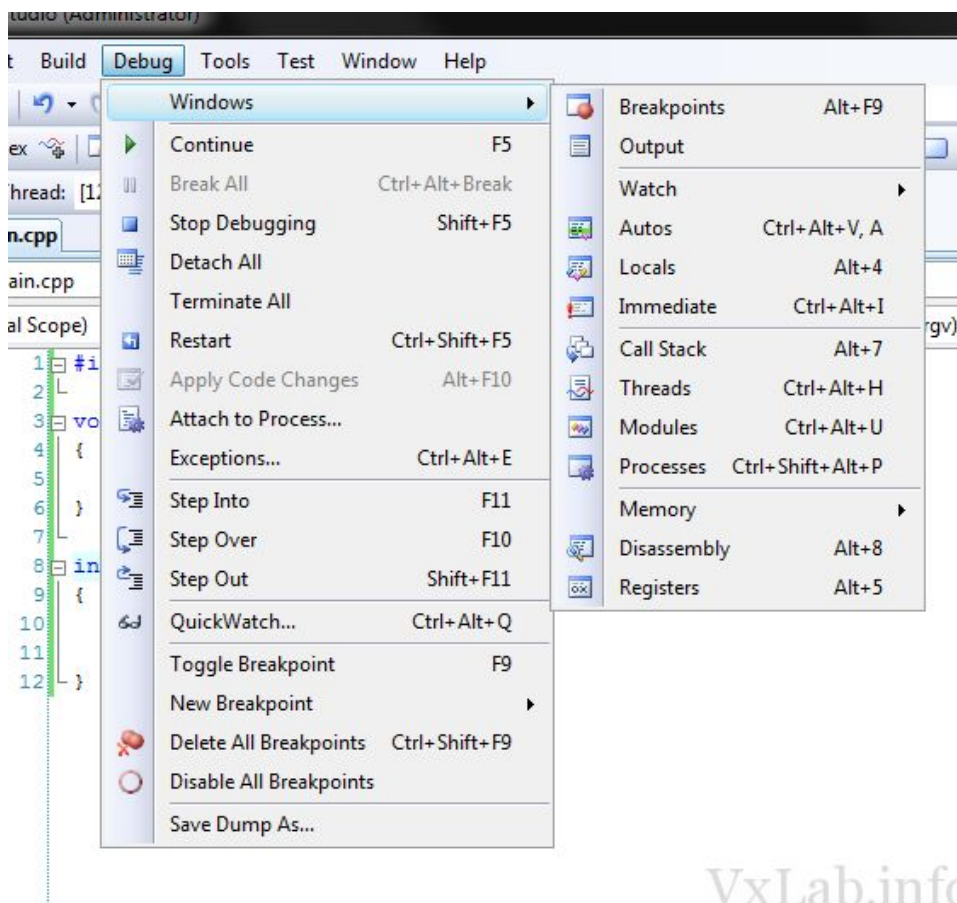


Работа с выводом отладчика

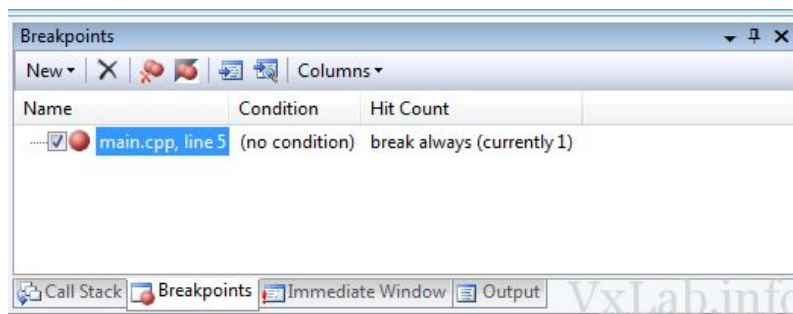
Запуск отладчика происходит при запуске проекта в среде разработки. При наличии установленных точек останова, выполнение прервется на первой из них автоматически. Важно понимать, что отладчик будет останавливать выполнение модуля напротив каждой активной точки останова. Подробнее поговорим о том, какая информация может быть доступна во время отладки. Прежде всего, информация эта четко структурирована и выводится в отдельных окнах.



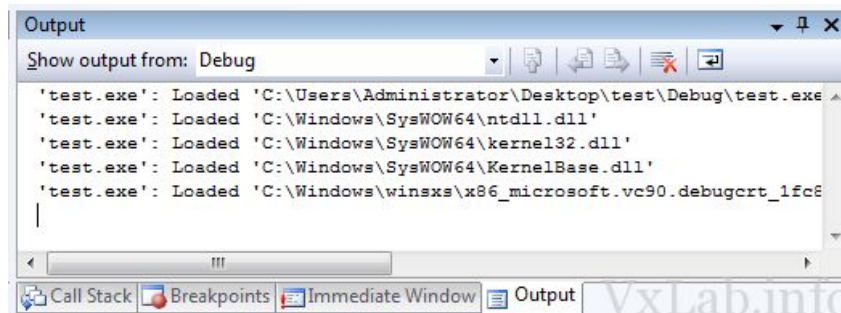
Посмотреть вывод всей возможной информации можно во вкладке Debug->Window.



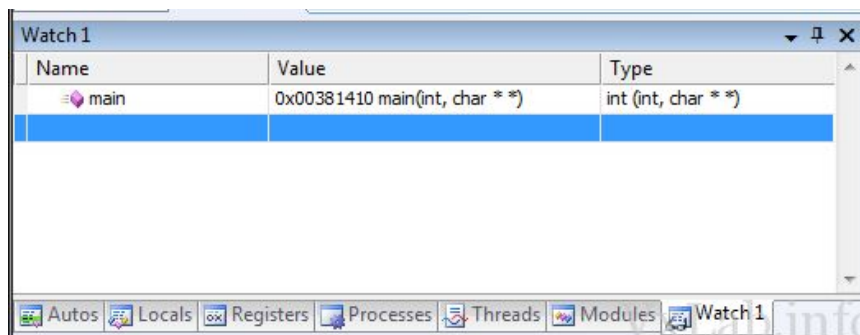
Breakpoints – информация обо всех точках остановки в отлаживаемом проекте.



Output – окно вывода Visual Studio. Используется для вывода служебных сообщений при работе со средой.

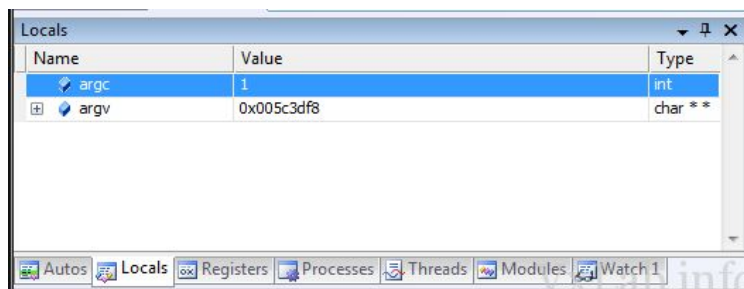


Watch – список наблюдаемых переменных. Переменные для наблюдения вносятся в список вручную и находятся там всегда, пока их не удалит разработчик.

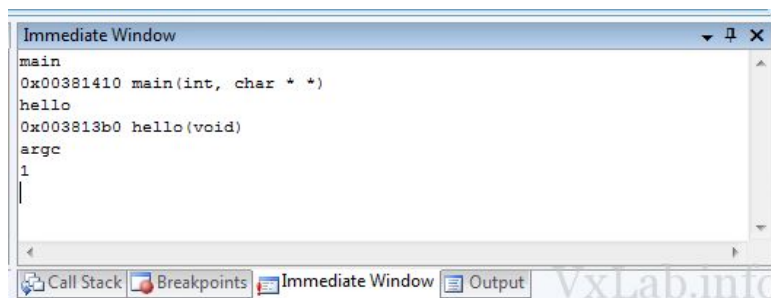


Autos – переменные, с которыми идет работа в данный момент. Т.е. актуальные на момент исполнения кода переменные и их значения.

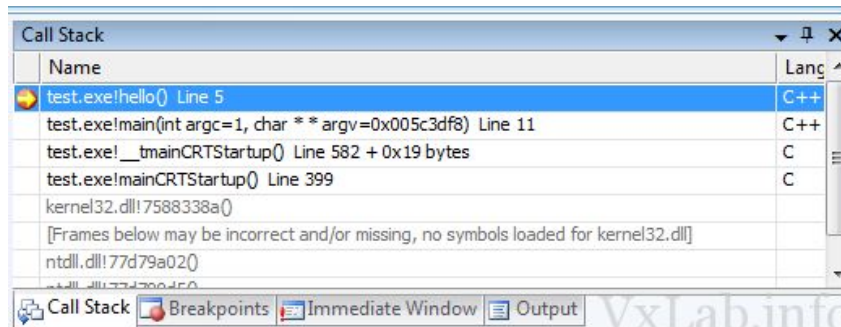
Locals – вывод локальных переменных и их значений.



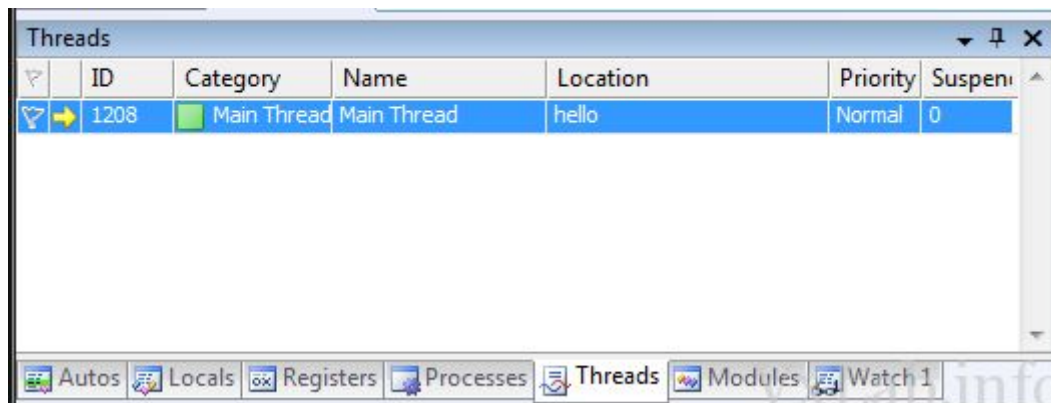
Immediate – поиск и вывод значения переменной по символьному представлению. В отличие от **Watch**, не хранит список значений, а выводит значение по требованию.



Call Stack – стек вызовов. Показывает последовательность вызовов функций для отлаживаемого модуля до точки останова.



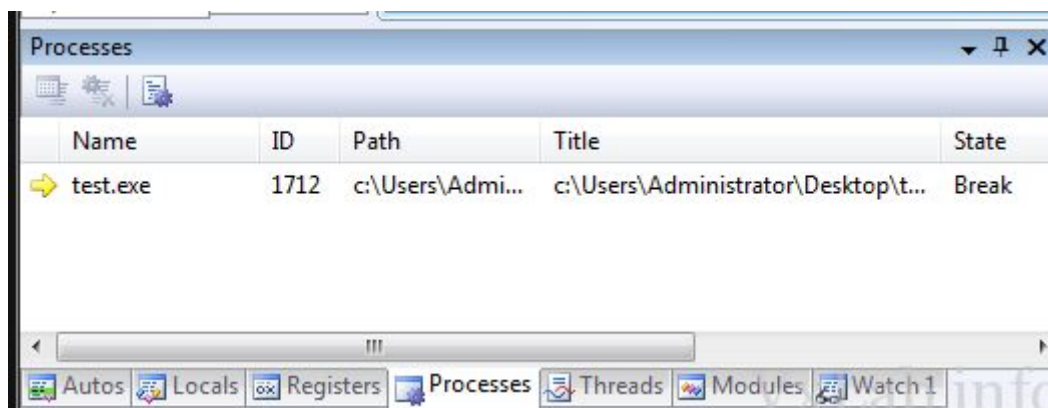
Threads – вывод информации обо всех запущенных потоках отлаживаемого модуля. Из этого окна возможна работа с потоками через контекстное меню: остановка потоков, переход к исходному коду потока и т.д.



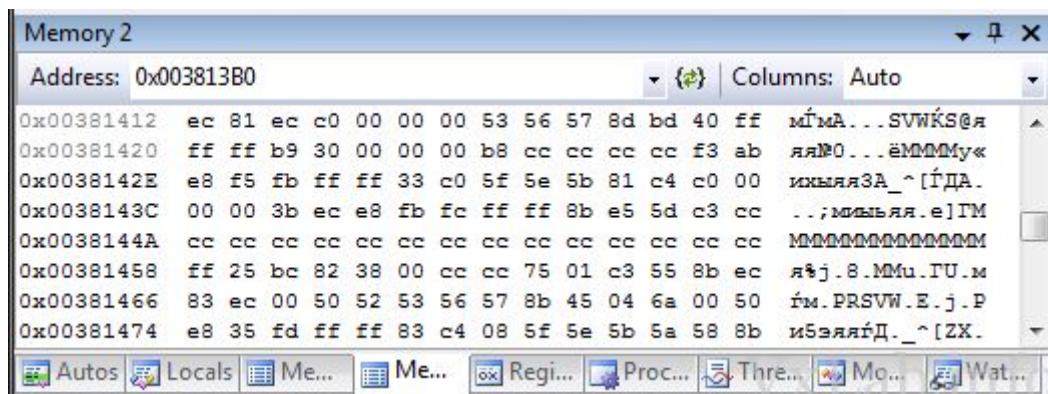
Modules – список загруженных модулей, необходимых для работы отлаживаемого (которые он подгружает для работы).



Processes – список процессов отлаживаемого модуля.



Memory – просмотр памяти отлаживаемого модуля в традиционном для любого hex-редактора виде.



Disassembly – просмотр ассемблерного листинга для отлаживаемого модуля. Очень хорошая возможность прямо из среды разработки посмотреть, как в итоге выглядит код на высокоуровневом языке.

Disassembly main.cpp

Address: hello

```

#include <stdio.h>

void hello()
{
    003813B0  push     ebp
    003813B1  mov      ebp,esp
    003813B3  sub      esp,0C0h
    003813B9  push     ebx
    003813BA  push     esi
    003813BB  push     edi
    003813BC  lea      edi,[ebp-0C0h]
    003813C2  mov      ecx,30h
    003813C7  mov      eax,0CCCCCCCCh
    003813CC  rep stos dword ptr es:[edi]
    printf("Hello, world");
    003813CE  mov      esi,esp
    003813D0  push     offset string "Hello, world" (38573Ch)
    003813D5  call     dword ptr [__imp__printf (3882BCh)]
    003813DB  add      esp,4
    003813DE  cmp      esi,esp
    003813E0  call     @ILT+315(__RTC_CheckEsp) (381140h)
}
    003813E5  pop      edi
    003813E6  pop      esi
    003813E7  pop      ebx
    003813E8  add      esp,0C0h
    003813EE  cmp      ebp,esp
    003813F0  call     @ILT+315(__RTC_CheckEsp) (381140h)
    003813F5  mov      esp,ebp
    003813F7  pop      ebp
    003813F8  ret

```

--- No source file -----

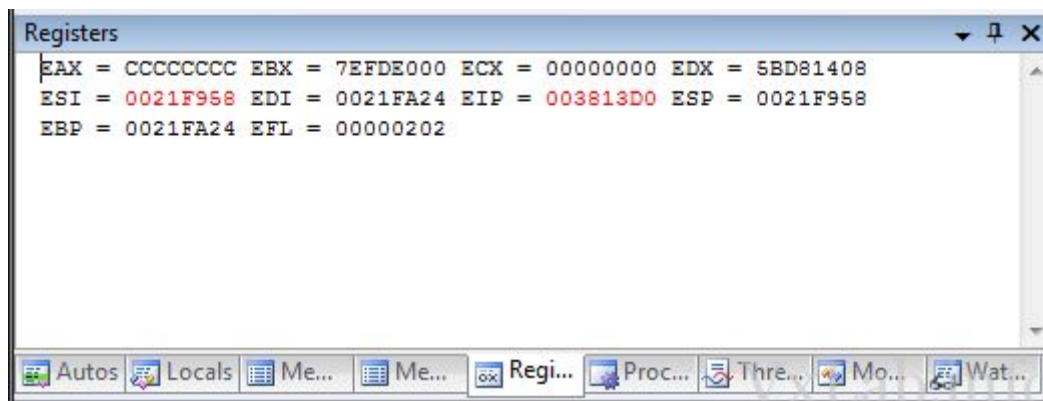
Registers – вывод значений в регистрах

Registers

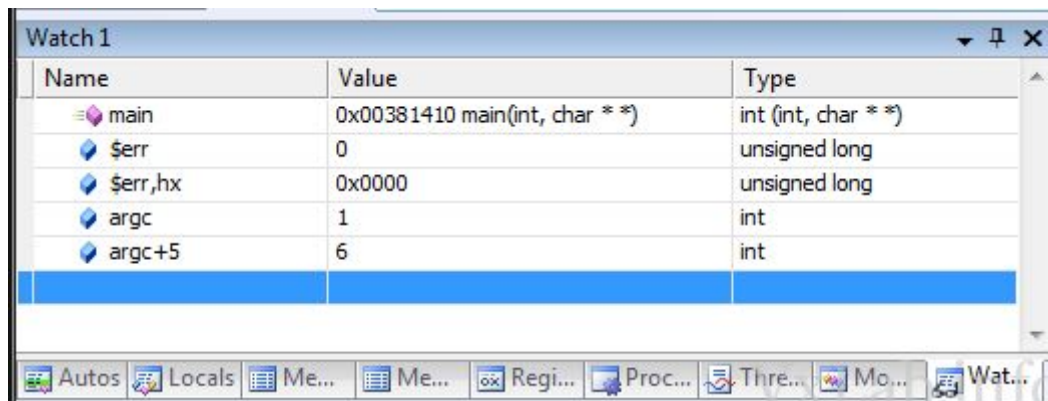
EAX = CCCCCCCC	EBX = 7EFDE000	ECX = 00000000	EDX = 5BD81408
ESI = 00000000	EDI = 0021FA24	EIP = 003813CE	ESP = 0021F958
EBP = 0021FA24	EFL = 00000202		

Что следует знать при работе с выводом отладчика

Информация в большинстве окон обновляется для каждой инструкции. Т.е. отладчик регистрирует и изменяет значения на новые. При этом измененные значения помечаются красным цветом.



Окно **Watch** поддерживает так называемые *псевдопеременные*. Своего рода макросы для определенных значений. Например, для вывода результата GetLastError после каждой строчки кода. Подробнее об этом можно прочитать [здесь](#). Также Watch выполняет арифметические операции с переменными, а для *managed*-кода может выполнять код.



Пошаговая отладка

Для того чтобы начать процесс отладки, необходимо поставить точку остановки в нужном месте и запустить модуль. Отладка станет доступна по достижению точки остановки. Также можно принудительно остановить выполнение модуля, в таком случае курсор автоматически переместится на код, который выполнялся до остановки. Но точка остановки при этом не установится.

Во время отладки в панели инструментов **Visual Studio** доступен Debug Toolbar. Если его нет, включить его можно, отметив элемент меню Debug во вкладке View->Toolbars.



Все кнопки в панели инструментов повторяют вкладку Debug, нас в данный момент интересует навигация по отлаживаемому коду. В качестве примера рассмотрим следующую ситуацию:


```
(Global Scope)
1 #include <stdio.h>
2
3 char* get_message()
4 {
5     return "Hello, world!";
6 }
7
8 void hello(char* message)
9 {
10    printf(message);
11 }
12
13 int main(int argc, char **argv)
14 {
15    hello(get_message());
16    return 0;
17 }
```

VxLab.info



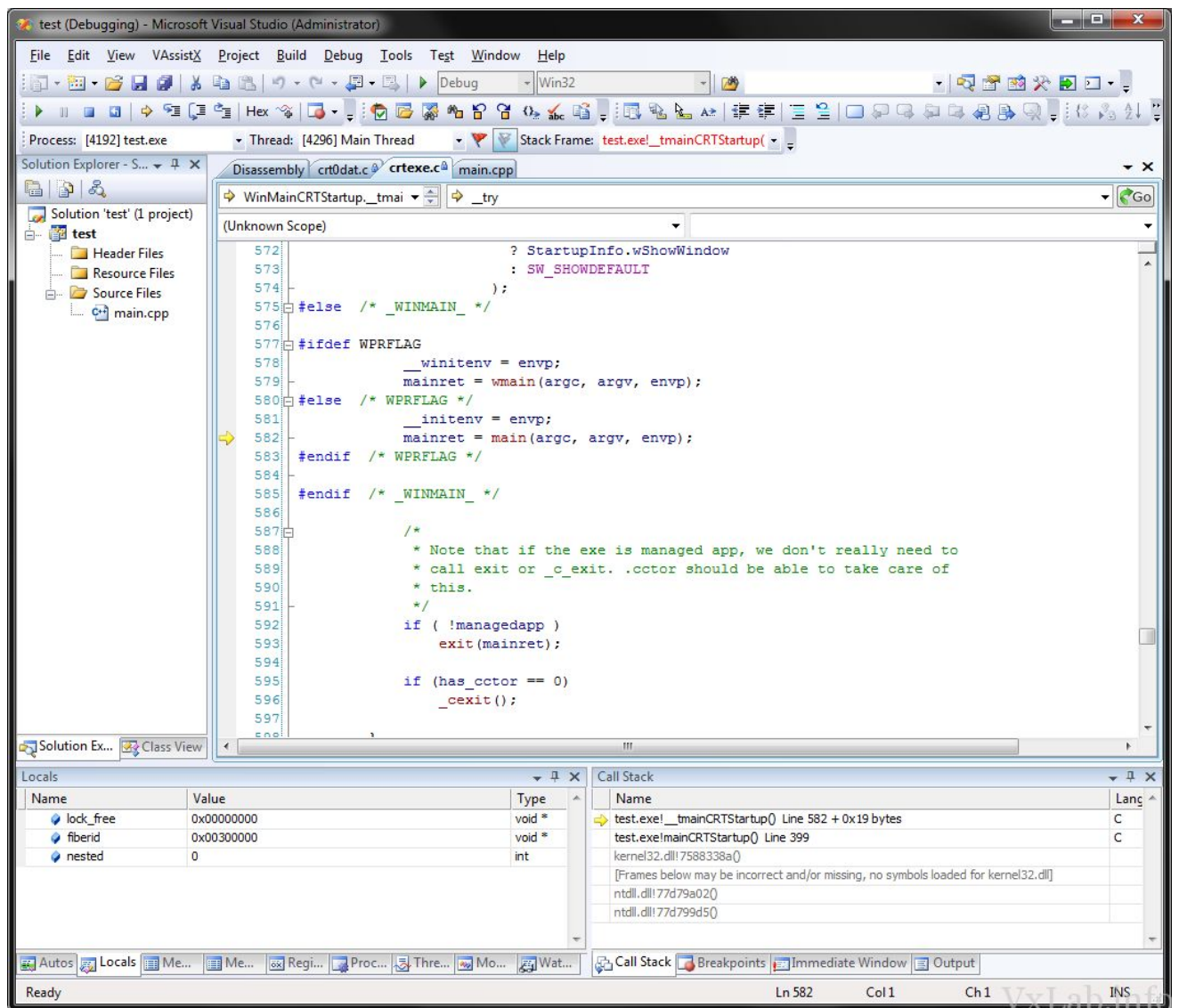
шаг со входом. (f11) Если отладка остановилась на вызове функции, шаг со входом необходим для того, чтобы остановить выполнение модуля внутри тела функции. Т.е. если внутри функции нет точки остановки, можно продолжить пошаговое выполнение, выполнив шаг со входом. Обратите внимание, в данном примере шаг со входом остановит выполнение программы внутри функции `get_message`, а не `hello`, т.к. сначала выполняется она. Т.е. шаг со входом в данном случае остановит выполнение модуля на строке 5.



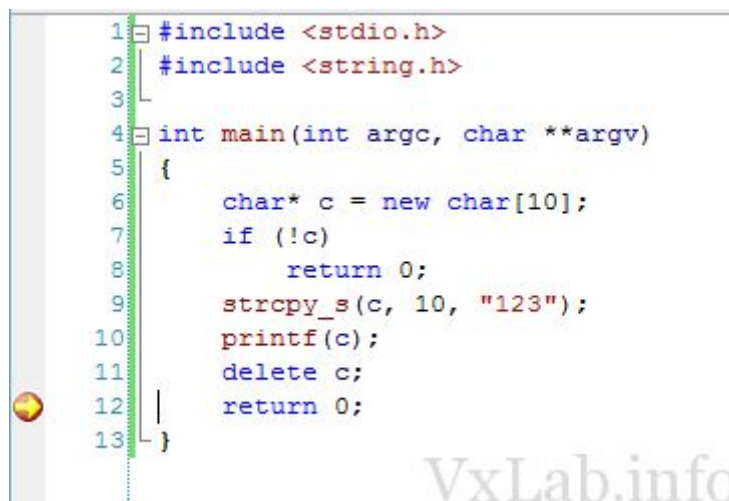
шаг с обходом. (f10) Пошаговая отладка внутри функции. Не переходит внутрь функции при ее вызове, а выполняет как обычную инструкцию. При завершении функции останавливается на верхней в стеке вызовов. Вне зависимости от того, сколько функций вызывается на строчке, отладчик не прерывает внутри этих функций выполнение модуля. В примере следующий шаг будет на строке 16.



шаг с выходом. (Shift + f11) В этом случае отладчик остановит выполнение модуля после выхода из текущей функции, где происходит отладка. В данном примере будет выход из функции `main` и переключение на отладку кода [CRT](#).



Во время пошаговой отладки важно также знать, что курсор можно передвинуть на необходимую строчку кода (Drag&Drop). Т.е. при желании заново отладить алгоритм не нужно перезапускать процесс отладки. Правда следует понимать, что в этом случае ожидаемое поведение программы не гарантировано. Разработчик должен отдавать отчет своим действиям. Например, в случае ниже программист получит ошибку, если передвинет курсор на строку 9, т.к. память для с еще не была выделена.



```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main(int argc, char **argv)
5 {
6     char* c = new char[10];
7     if (!c)
8         return 0;
9     strcpy_s(c, 10, "123");
10    printf(c);
11    delete c;
12    return 0;
13 }

```

Основные команды меню Build

Compile — компиляция выбранного файла. Результаты компиляции выводятся в окно Output.

Build — компоновка проекта. Компилируются все файлы, в которых произошли изменения с момента последней компоновки. После компиляции происходит сборка (link) всех объектных модулей, включая библиотечные, в результирующий исполняемый файл. Сообщения об ошибках компоновки выводятся в окно Output. Если обе фазы компоновки завершились без ошибок, среда программирования создаст исполняемый файл с расширением *.exe (для данного примера: lr1.exe), который можно запустить на выполнение.

Rebuild All — то же, что и Build, но компилируются все файлы проекта независимо от того, были ли в них произведены изменения или нет.

Execute — выполнение исполняемого файла, созданного в результате компоновки проекта. Если же в исходный текст были внесены изменения – то перекомпилирование, перекомпоновку и выполнение.

Операции **Compile**, **Build** и **Execute** соответственно первая, вторая и четвертая кнопки панели инструментов **Build MiniBar**, которая расположена на рабочем столе (рисунок 3) справа вверху рядом с системными кнопками.

Управление конфигурациями проекта в Visual Studio 2010

Любой проект в VisualStudio2010 включает несколько самостоятельных конфигураций для компиляции разных версий программы. Конфигурацией называется набор параметров компилятора, компоновщика и библиотекаря, используемый при построении проекта. По умолчанию при создании проекта среда VisualStudio2010 автоматически включает в него две конфигурации: **Debug**(отладочную) и **Release**(финальную). Как следует из их названий, отладочная конфигурация используется в процессе написания программы, ее тестовых

запусков для обнаружения и исправления ошибок, финальная – 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** конфигурация компилируется с включением полной символьной отладочной информации и выключением оптимизации. 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, т. е. для поддержки стандартных действий, выполняемых при отладке программы.
- **Release** конфигурация не содержит никакой отладочной информации и подвергается полной оптимизации. Без отладочной информации процесс отладки программы очень затруднен. Однако при необходимости такая информация может быть создана 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 ConfigurationManager(менеджер конфигураций). Для быстрого переключения конфигурации, используемой для компиляции проекта, используется стандартная панель инструментов (рисунок П1).

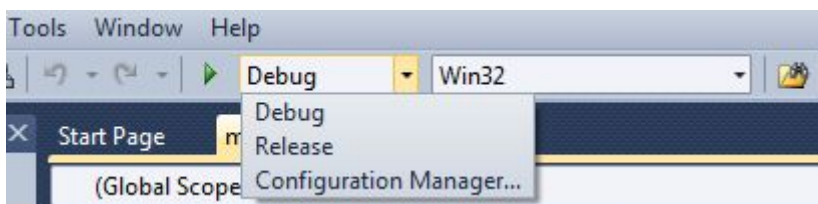


Рисунок П1 – Переключение конфигураций из панели инструментов

Для того чтобы проверить какая текущая конфигурация в проекте, нужно выбрать в подменю Project пункт Settings. Откроется диалоговое окно Project Settings. Смотрим, какое значение установлено в окне комбинированного списка Settings For:. Если это не Win32 Debug, то переключитесь на нужное значение через команду меню Build/Set Active Configuration

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Создать проект для кода C, осуществить все необходимые настройки интегрированной среды. Набрать код программы C.

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

```

#include <conio.h>

int summa(int a, int b){
    printf("Work function summa\n");
    return a+b;
}

int mult(int a, int b){
    printf("\nWork function multiplication");
    return a*b;
}

int fact(int a){
    int f=1;
    for(int i=1; i<a+1; i++ ) {

        f*=i;
    }
    printf("\nWork function factorial");
    return f;
}

int main( )
{
    int a,b,s,p;
    printf("Enter the value of the variable a=");
    scanf("%d", &a);
    printf("\nEnter the value of the variable b=");
    scanf("%d",&b);
    s=summa(a,b);
    printf("The sum of two numbers (a+b)=%d", s);
    p=mult(a,b);
    printf("\nThe multiplication of two numbers (a*b)=%d",p);
    printf("\nEnter the value of the variable f=");
    scanf("%d",&p);
    printf("\nFactorial number %d equal %d",p,fact(p));
    getchar();
    scanf ("Vvod s=",&s);
    return 0;
}

```

2. Создать проект для кода C++, осуществить все необходимые настройки интегрированной среды. Набрать код программы C++.

```

#include <iostream>
#include <conio.h>
using namespace std;
int summa(int a, int b){
    cout<<"Work function summa\n";
    return a+b;
}
int mult(int a, int b){
    cout<<"\nWork function multiplication";
    return a*b;
}
int fact(int a){
    int f=1;
    for(int i=1; i<a+1; i++ ) {
        f*=i;
    }
    cout<<"\nWork function factorial";
    return f;
}

```

```

int main( )
{
    int a,b,s,p;
    cout<<"Enter the value of the variable a=";
    cin>>a;
    cout<<"Enter the value of the variable b=";
    cin>>b;
    s=summa(a,b);
    cout<<"The sum of two numbers (a+b)= "<<s;
    p=mult(a,b);
    cout<<"\nThe multiplication of two numbers (a*b)= "<<p;
    cout<<"\nEnter the value of the variable f=";
    cin>>p;
    cout<<"Factorial number  "<<p<<"    equal  "<<fact(p)<<endl;
    cout<<endl;
    getchar();
    system("PAUSE");
    return 0;
}

```

3. Выполнить программу C++ пошагово, используя клавишу F10. Результат занести в отчет. Пояснить в чем особенность данного режима отладки. Выводы занести в отчет.
4. Выполнить программу C++ пошагово, используя клавишу F11. Результат привести в отчете. Пояснить в чем особенность данного режима отладки. Выводы занести в отчет.
5. Создать 2 точки останова на выделенных строках в коде программы C++.

```

cout<<"Enter the value of the variable b=";
cin>>b;
s=summa(a,b);
cout<<"The sum of two numbers (a+b)= "<<s;
p=mult(a,b);
cout<<"\nThe multiplication of two numbers (a*b)= "<<p;

```

Выполнить программу по нажатию на F5 до строки

```
cout<<"Enter the value of the variable b=";
```

Со строки

```
cout<<"Enter the value of the variable b=";
```

(выполнить программу пошагово по нажатию клавиши F10)

до строки

```
cout<<"\nThe multiplication of two numbers (a*b)= "<<p;
```

После строки

```
cout<<"\nThe multiplication of two numbers (a*b)= "<<p;
```

завершить выполнение программы по нажатию на F5.

В отчете пояснить полученный результат. Пояснить, для чего нужны точки останова.

6. Удалить точки останова в коде C++.
7. Научиться анализировать данные в окне **Output**, научиться добавлять переменные в окно отладки. Занести переменные a и b в окно отладки и проследить, как изменяются их значения, результаты занести в отчет.
8. Выполните компиляцию проекта C++ в отладочной конфигурации (Win32 Debug). Поясните назначение данной отладочной конфигурации. Посмотрите состав файлов в проекте. Поясните назначение каждого файла. Занесите результаты в отчет. Занесите размер проекта в отчет.
9. Выполните компиляцию проекта C++ в отладочной конфигурации (Win32 Release). Поясните назначение данной отладочной конфигурации. Посмотрите состав файлов в

проекте. Поясните назначение каждого файла. Занесите результаты в отчет. Занесите размер проекта в отчет.

10. Проведите анализ размеров и состав файлов проекта C++ в конфигурациях (Win32 Debug) и (Win32 Release). Результаты занесите в отчет.
11. Получите исполняемый файл, созданный в результате компоновки проекта C++. Посмотрите его место размещения на диске. Запустите его и посмотрите результат. Занесите результаты в отчет.