

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ**

Кафедра автоматизированных систем управления (АСУ)

М.Ю. Катаев

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лабораторный практикум

2014

Корректор: Осипова Е.А.

Катаев М.Ю.

Объектно-ориентированное программирование: лабораторный практикум. — Томск: Факультет дистанционного обучения, ТУСУР, 2014. — 53 с.

© Катаев М.Ю., 2014

© Факультет дистанционного
обучения, ТУСУР, 2014

СОДЕРЖАНИЕ

1 Изучение среды разработки Visual C++ 6.0.....	4
1.1 Пользовательский интерфейс Visual C++ 6.0.....	6
1.2 Меню и панели инструментов.....	6
1.3 Настройка параметров среды	10
1.4 Система помощи приложения.....	11
1.5 Параметры конфигурации проекта.....	12
1.6 Типы мастеров проектов.....	13
1.7 Создание проекта VC++.....	15
1.8 Выбор типа проекта.....	15
1.9 Добавление файлов и классов в проект	18
1.10 Создание классов посредством мастера	23
1.11 Добавление полей и методов мастером	25
2 Среда разработки Visual C++ 2005	30
2.1 Пользовательский интерфейс.....	30
2.2 Меню и панели инструментов.....	32
2.3 Настройка параметров среды	35
2.4 Параметры конфигурации проекта.....	36
2.5 Типы мастеров проекта.....	37
2.6 Выбор типа проекта.....	39
2.7 Добавление файлов в проект	40
2.8 Создание классов по средством мастера.....	41
2.9 Пример программы вывода текста на экран.....	44
3 Лабораторные работы по объектно-ориентированному программированию	45
3.1 Работа № 1	47
3.2 Работа № 2	48
4 Правила оформления лабораторных работ	53

1 ИЗУЧЕНИЕ СРЕДЫ РАЗРАБОТКИ VISUAL C++ 6.0

Создание приложений для работы в операционной среде Windows на языке Visual C++ включает следующие этапы:

- Написание текстов программ.
- Создание файлов ресурсов для описания диалоговых окон, меню, панелей инструментов, значков и т. п. — элементов управления работой программы.
- Разработка системы помощи для работы с программой.
- Компилирование исходных текстов программ.
- Компоновка программы их компонентов.
- Отладка и модификация программ.

Опишем некоторые, наиболее популярные, бесплатные компиляторы C++, знание которых ускорит процесс освоения среды разработки программ. Можно отметить, что Microsoft Visual C++ компилятор является не единственной интегрированной средой разработки программ C++, которые популярны и широко используются программистами. Выделим небольшую часть компиляторов, среда разработки которых удобна для пользования начинающими программистов C++.

1. Microsoft Visual C++ Express (<http://msdn.microsoft.com/vstudio/express/visualc/>) — бесплатный компилятор для Windows и .NET.

2. Eclipse C++ Development Tools [<http://www.eclipse.org/cdt/>] — плагин разработки C++ для Eclipse. Работает с MinGW32, хотя можно подключать и другие компиляторы. Удобный интерфейс, средства для коллаборации разработчиков и UML-разработки присутствуют.

3. Dev C++ [<http://www.bloodshed.net/devcpp.html>] — вторая по популярности среда разработки для C++. Поддерживает GCC компиляторы, подсветку, auto-complete, шаблоны, профайлинг, дебаггинг, class-view и прочий стандартный набор для такого рода программ. Интерфейс удобный, присутствуют средства для работы с CVS.

4. Code::Blocks [<http://www.codeblocks.org/>] — бесплатная среда разработки под C++.

5. Digital Mars [<http://www.digitalmars.com/>].

6. Visual GWin++ [<http://sourceforge.net/projects/visual-gwin/files/>].

7. MinGW Developer Studio [<http://www.mingw.org/>] — программа для Windows (Minimalist GNU for Windows). По существу это порт GNU Compiler Collection (GCC) для Windows плюс бесплатная коллекция, включающая типовые заголовочные файлы и библиотеки.

8. C++ Builder [<http://www.embarcadero.com/ru/products/cbuilder>] — среда для действительно быстрой разработки приложений на C++. Огромное количество возможностей.

9. KDevelop [<http://kdevelop.org/>] — стабильная среда разработки Си++/Си, даёт хорошую интеграцию с инструментами QT [<http://qt-project.org/>].

10. Open Watcom [http://www.openwatcom.org/index.php/Main_Page] — проект сообщества открытого кода по поддержке и развитию многоплатформных компиляторов Watcom C, C++ и Fortran и сопутствующих программ.

Каждый из указанных выше компиляторов позволяет разрабатывать программы C++, однако часть из них может быть использована лишь для разработки небольших по объёму программ. Предлагается читателям воспользоваться ссылками и протестировать компиляторы, выбрать наиболее удобный и понятный для себя компилятор. Однако для практики, конечно же, важным и необходимым является компилятор Microsoft Visual C++.

1.1 Пользовательский интерфейс Visual C++ 6.0

Приведем внешний вид окна рабочего пространства среды Visual C++ 6.0. Рабочая область разделена на две части: окно рабочей области и окно редактирования.

Для графического отображения объектов рабочей области используется список с древовидным отображением, корневыми узлами которого являются проекты. Содержимое этих проектов можно представить тремя способами, каждому из которых соответствует вкладка в нижней левой части рабочей области:

- **ClassView** — представляет программу в объектно-ориентированном виде, отображая классы C++, их методы и члены-данные. Двойной щелчок такого объекта вызовет переход к его объявлению или реализации;
- **ResourceView** — отображает ресурсы, сгруппированные по категориям. Двойной щелчок объекта загружает соответствующий редактор;
- **FileView** — показывает все файлы проекта, которые можно редактировать.

Если щелкнуть объект правой кнопкой мыши, появится контекстное меню, пункты которого связаны с этим объектом.

Для работы с файлами в редакторах программ и ресурсов используется *окно редактирования*. Если Вам не хватает в нем пространства, выберите из меню **View** опцию **FullView**.

1.2 Меню и панели инструментов

Среда разработки Visual C++ обладает набором меню, позволяющим управлять файлами и рабочими областями проектов, настраивать саму среду, а также обращаться к справочной системе, программе управления исходным кодом и внешним инструментальным средствам. Почти каждому меню соответствует панель инструментов, в которой команду можно выбрать одним щелчком кнопки мыши. Панели инструментов допустимо настраивать — добавлять и удалять кнопки, скрывать и отображать сами панели инструментов. Поэтому вы можете сконфигу-

рировать среду по своему вкусу, упростив доступ к часто используемым вами пунктам меню.

По умолчанию в окне среды Visual C++ отображены три панели инструментов. Панель **Standart** (содержит команды, часто используемые при работе с файлами), **Build** (команды, необходимые для сборки и запуска приложений) и **WizardBar** (представляет инструменты для работы с классами).

При щелчке правой кнопки мыши на поле расположения панелей (например, в пустой правой верхней части панели инструментов) появится полный список возможных панелей, позволяющий скрывать и отображать панели.

Показанное ниже упражнение позволит Вам получить опыт в настройке пользовательского интерфейса.

Задача: добавление кнопки в панель инструментов:

1. В меню **Tools** выберите пункт **Customize**. Появится диалоговое окно **Customize**.

2. Перейдите на вкладку **Commands**.

3. В списке **Category** выберите пункт **View**.

4. Щелкните значок **Full Screen** (третий слева в первом ряду). Обратите внимание на описание команды, которое появляется под раскрывшимся списком **Category**.

5. Перетащите значок **Full Screen** на панель инструментов **Standart**.

6. Закройте окно **Customize**.

7. Откройте файл с исходным кодом программы, дважды щелкнув его в окне **FullView**.

8. Испытайте новую кнопку — щелкните ее, чтобы перейти в полноэкранный режим. Чтобы вернуться к традиционному виду, необходимо снова щелкнуть эту кнопку.

Чтобы удалить кнопку с панели инструментов, достаточно щелкнуть ее правой кнопкой мыши и выбрать из контекстного меню пункт **Delete**.

Помимо жестко установленных на панели кнопок, можно определять так называемые «быстрые» клавиши. Рассмотрим алгоритм установки такого объекта.

1. В меню **Tools** выберите пункт **Customize**. Появится диалоговое окно **Customize**.

2. Щелкните вкладку **Keyboard**.
3. В раскрывшемся списке **Category** выберите опцию **View**.
4. В списке **Commands** укажите **ToggleFullScreen**. Обратите внимание на описание команды, появившееся под раскрывшимся списком **Category**.
5. Щелкните после ввода **Press new shortcut key**, после чего нажмите на клавишу PAUSE.
6. Щелкните кнопку **Assign**.
7. Закройте диалоговое окно **Customize**.
8. Нажмите клавишу PAUSE. При этом окно редактирования должно переключаться из оконного в полноэкранный режим и обратно.

Рассмотрим некоторые, самые часто используемые, команды меню. Подменю File служит для работы с проектом в целом, а также с его файлами.

Команда File $\Rightarrow$ New вызывает диалоговое окно для создания нового файла (подопция Files), проекта (подопция Projects), рабочего пространства (подопция Workspaces) или других документов (подопция Other Documents).

Команда File $\Rightarrow$ Open позволяет открыть существующий на жестком диске (или дискете или СД) файл.

Команда File $\Rightarrow$ Close закрывает файл, имеющий фокус ввода (расположение курсора мыши).

Команда File $\Rightarrow$ Open Workspaces открывает существующий файл с описанием рабочей области.

Команда File $\Rightarrow$ Save Workspaces запоминает изменения в рабочей области.

Команда File $\Rightarrow$ Close Workspaces вызывает закрытие рабочей области с одновременным сохранением изменений.

Команда File $\Rightarrow$ Save (Save as, Save All) позволяет сохранить файл, сохранить файл с новым именем или сохранить несколько файлов одновременно.

Команда File $\Rightarrow$ Page Setup выводит диалоговое окно для установки размеров верхнего и нижнего колонтитулов и полей страницы.

Команда File $\Rightarrow$ Print позволяет из соответствующего диалогового окна распечатывать указанный диапазон страниц.

Команда File $\Rightarrow$ Recent Files (Recent Workspaces) выводит список файлов и рабочих областей, которые использовались последними, для их быстрого открытия.

Команда File $\Rightarrow$ Exit завершает работу с Visual C++.

Подменю Edit служит для редактирования текстов, поиска и замены, задания закладок и точек остановки для отладки программы.

Команда Edit $\Rightarrow$ Undo отменяет операцию, выполненную последней. Повторное нажатие вызывает отмену предыдущей операции.

Команда Edit $\Rightarrow$ Redo восстанавливает последнюю операцию.

Команда Edit $\Rightarrow$ Cut переносит выделенный участок файла в буфер обмена с удалением выделенного из самого текста файла.

Команда Edit $\Rightarrow$ Copy копирует выделенный участок файла в буфер без его удаления из текста.

Команда Edit $\Rightarrow$ Paste копирует содержимое буфера в то место файла, где установлен курсор.

Команда Edit $\Rightarrow$ Delete удаляет выделенный участок без копирования в буфер.

Команда Edit $\Rightarrow$ Select all выделяет весь файл.

Команда Edit $\Rightarrow$ Find (Find in Files) открывает диалоговое окно, в котором необходимо указывать, какой текст следует искать и в каком направлении (от начала к концу файла или наоборот).

Команда Edit $\Rightarrow$ Replace производит замену одних выражений другими.

Команда Edit $\Rightarrow$ Go To позволяет перейти к адресу, полученному при выполнении отладки, к закладке, к строке с заданным номером, к следующей ошибке, найденной при компиляции программы.

Команда Edit $\Rightarrow$ Bookmarks устанавливает именованные закладки для быстрого перемещения по тексту программы.

Команда Edit $\Rightarrow$ Advanced активизирует меню третьего уровня со следующими командами:

- Increment Search — производит поиск по мере набора текста для поиска,
- Format — устанавливает отступы в выделенном участке текста,
- Tabify Selection (Untabify) — преобразует символы табуляции в пробелы и обратно,
- Make Selection Uppercase (Lowercase) — переводит выделенные символы табуляции в пробелы и обратно,
- Whitespace — вставляет символы-заполнители для знаков табуляции и пробелов.

Команда Edit $\Rightarrow$ Breakpoints позволяет установить точки останова для выполнения отладки программы.

Команда Edit $\Rightarrow$ List Members позволяет вывести полный список членов класса или структуры после набора оператора доступа к его члену.

Команда Edit $\Rightarrow$ Type Info выводит описание любого идентификатора программы, на котором находится курсор.

Команда Edit $\Rightarrow$ Parameter Info выводит описание аргументов функции при наборе ее имени и открывающейся скобки.

Команда Edit $\Rightarrow$ Complete Word позволяет автоматически завершить набор слова, если в списке доступных слов имеется единственное слово, имеющее начало, которое уже подобрано.

1.3 Настройка параметров среды

Выбрав в меню Tools пункт Options, можно вызвать диалоговое окно Options. В этом окне настраиваются параметры Вашей рабочей среды, от которых зависит вид и возможности встроенных в нее инструментальных средств. Данные параметры применяются ко всем проектам и их конфигурациям.

В диалоговом окне можно настроить:

1) редактор кода — начиная с параметров окна и сохранения и заканчивая параметрами таких функций, как автозаполнение кода и цветовое выделение синтаксических элементов. Можно эмулировать некоторые версии редакторов;

2) отображение информации отладчиком;

3) параметры встроенной программы управления исходным кодом;

4) пути к исполняемым файлам, файлам с исходным кодом, включаемым файлам и библиотекам.

1.4 Система помощи приложения

Visual Studio 6.0 поставляется вместе с библиотекой **MSDN** (Microsoft Developer Network). Это справочник разработчика, содержащий более 1 Гб технической информации по всем сторонам программирования (включены: документация, технические статьи, образцы кода и много другой информации о программировании на продуктах фирмы Microsoft). Библиотека **MSDN** запускается в собственной среде, основанной на **HTML** и снабженной быстрой поисковой системой. Несмотря на внешнюю изолированность, программа **MSDN** интегрирована в среду разработки Visual C++.

В Windows существуют три типа сообщений, посылаемых в тот момент, когда пользователь обращается к справочной системе:

- 1) WM_COMMAND;
- 2) WM_HELP;
- 3) WM_CONTEXTMENU.

Когда пользователь выбирает в меню команду Help, система посылает сообщение WM_COMMAND. Для отображения соответствующей справки вы должны перехватить это сообщение и вызывать систему WinHelp.

Когда пользователь щелкает правой кнопкой мыши на некотором элементе программы, посылается сообщение WM_CONTEXTMENU. Надо перехватить это сообщение и вывести его в данном месте контекстного меню.

*.h	Файлы заголовков содержат определения идентификаторов ресурсов и идентификаторы тем справок, которые будут использоваться в программе
*.hm	Файлы адресации справок содержат идентификаторы тем справки. Этот файл генерируется всякий раз, когда компилируется приложение (не рекомендуется вносить в него изменения)

*.rtf	Файлы расширенного текстового формата содержат тексты справок по каждой теме
Name.cnt	Файл таблицы содержания вкладки Contents (Содержание) в диалоговом окне Help Topics. Этот файл оглавления поставляется вместе с приложением в дополнение к файлу *.hlp
Name.hpj	Файл проекта справочной системы объединяет файлы *.hm и *.rtf, совместно используемые при компиляции файла *.hlp

Когда пользователь обращается к справочной службе любым другим способом (например, используя горячую кнопку F1), большая часть работы выполняется специально встроенным обработчиком событий. В данном случае перехватывается сообщение WM_HELP.

При использовании интерактивной справочной системы используется большое количество различных файлов. Конечным продуктом является файл, имеющий расширение *.hlp. Он создается на основе нескольких файлов. В приведенной ниже таблице, слово Name следует заменить на имя *.exe файла приложения. Известны следующие расширения файлов, служащих для разработки компонент справочной системы.

1.5 Параметры конфигурации проекта

При работе над проектом в некоторых случаях возникает необходимость изменения параметров конфигурации проекта. Для этого вызовите окно **Project Settings**, выбрав в меню **Project** пункт **Settings**. В этом окне можно настроить параметры проекта, в том числе и параметры компиляторов и компоновщика Visual C++.

Важно отметить, что изменения параметров на любой вкладке влияют только на текущий проект и версию сборки, указанные в раскрывающемся списке **Settings For**. Для каждого типа сборки можно задать свой набор параметров. Если требуется установить одинаковые параметры для всех версий сборки, в списке **Settings For** щелкните **All Configurations**.

Не менее важно при настройке проекта проверить, какую его конфигурацию Вы изменяете.

Приведем описание содержания некоторых вкладок диалогового окна **Project Settings**.

- **General Settings** — Здесь можно указать, компоновать ли программу с использованием статических библиотек MFC (если вы решили изменить способ компоновки, выбранной вами в окне мастера **AppWizard**). Также можно задать каталоги, в которые будут помещаться конечные файлы.
- **Debug Settings** — Здесь находятся параметры, передаваемые программе при ее запуске в отладчике. Кроме того, можно использовать переадресацию ввода/вывода, аналогичную применяемой в командной строке.
- **C/C++ Settings** — Здесь вы определяете параметры компилятора, функциональные возможности языка, соглашения о вызове, параметры, связанные с типом процессора, оптимизации кода, константами препроцессора, и т. п.
- **Linker Settings** — На этой вкладке выбирают дополнительные библиотеки, которые следует скомпилировать вместе с вашей программой.

1.6 Типы мастеров проектов

В среде Visual C++ можно строить различные типы проектов. Такие проекты после их создания можно компилировать и запускать на исполнение. Фирма Microsoft разработала специальный инструментарий, облегчающий и ускоряющий создание проектов в среде Visual C++.

Рассмотрим некоторые типы проектов, которые можно создавать при помощи различных средств (мастеров проектов) Microsoft Visual C++:

MFC AppWizard (exe) — при помощи этого мастера приложений можно разработать проект Windows-приложения, имеющего однодокументный, многодокументный или диалоговый интерфейс. Однодокументное приложение может предоставлять возможность пользователю в любой момент времени работать только с одним документом в окне. Многодокументное приложение, напротив, может одновременно представлять несколько документов, каждый в собственном окне. Пользовательский интерфейс диалогового приложения представляет собой единственное диалоговое окно.

MFC AppWizard (dll) — этот мастер приложений позволяет создать структуру DLL, основанную на MFC. При помощи него можно определить характеристики будущей DLL.

AppWizard ATL COM — это средство позволяет создать элемент управления ActiveX или сервер автоматизации, используя новую библиотеку шаблонов ActiveX (ActiveX Template Library — ATL). Опции этого мастера дают возможность выбрать активный сервер (DLL) или исполняемый внешний сервер (exe-файл).

Custom AppWizard — при помощи этого средства можно создать пользовательские мастера AppWizard. Пользовательский мастер может базироваться на стандартных мастерах для приложений MFC или DLL, а также на существующих проектах или содержать только определяемые разработчиком шаги.

DevStudio Add-in Wizard — мастер дополнений позволяет создавать дополнения к Visual Studio. Библиотека DLL расширений может поддерживать панели инструментов и реагировать на события Visual Studio.

MFC ActiveX ControlWizard — мастер элементов управления реализует процесс создания проекта, содержащего один или несколько элементов управления ActiveX, основанных на элементах управления MFC.

Win32 Application — этот мастер позволяет создать проект обычного Window-приложения. Проект создается незаполненным, файлы с исходным кодом в него следует добавлять вручную.

Win32 Console Application — мастер создания проекта консольного приложения. Консольная программа — это программа, которая выполняется из командной строки окна DOS или Windows и не имеет графического интерфейса (окон). Проект консольного приложения создается пустым, предполагая добавление файлов исходного текста в него вручную.

Win32 Dynamic-Link Library — создание пустого проекта динамически подключаемой библиотеки. Установки компилятора и компоновщика будут настроены на создание DLL. Исходные файлы следует добавлять вручную.

Win32 Static Library — это средство создает пустой проект, предназначенный для генерации статической (объектной) библиотеки. Файлы с исходным кодом в него следует добавлять вручную.

1.7 Создание проекта VC++

Самым распространенным в мире учебным и производственным пакетом программ является продукт фирмы Microsoft — Visual Studio 6.0, в состав которого входит среда программирования Visual C++. Несмотря на то, что в настоящее время уже существует новое поколение среды разработки Visual Studio.NET, VC++ версии 6.0 не потеряло своих возможностей.

При выполнении лабораторных работ мы будем использовать среду разработки Visual C++ 6.0. А так как эта среда требует определенных действий для создания программы, кроме написания исходного кода, то мы сейчас их и рассмотрим.

1.8 Выбор типа проекта

Итак, для написания программы нам необходимо создать проект (рабочее пространство). Для этого необходимо запустить Microsoft Visual C++ 6.0. После чего выполняем команду: File — New, или нажимаем сочетание клавиш Ctrl+N. Когда вы выполните эти действия, появится диалоговое окно, такое же, как приведено на рис. 1.

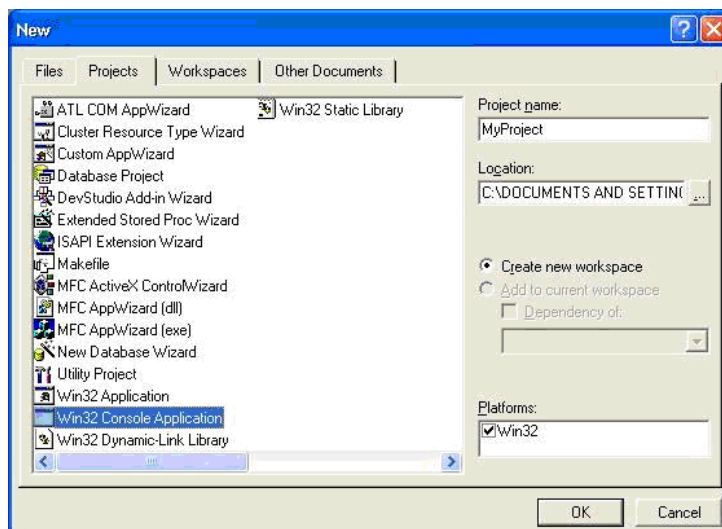


Рис.1 — Диалоговое окно выбора типа проекта

В этом окне выбираем тип проекта Win32 Console Application, так как пока будем работать с консольным проектом.

В поле Project name вводим название нашего проекта, в нашем случае это MyProject. Теперь нажимаем кнопку Ok. После этого появятся еще два диалоговых окна. В первом ничего не меняем, а просто нажимаем кнопку Finish, т.е. создадим пустой проект. Пример второго, возникающего при выборе проекта окна расположен на рис. 2.

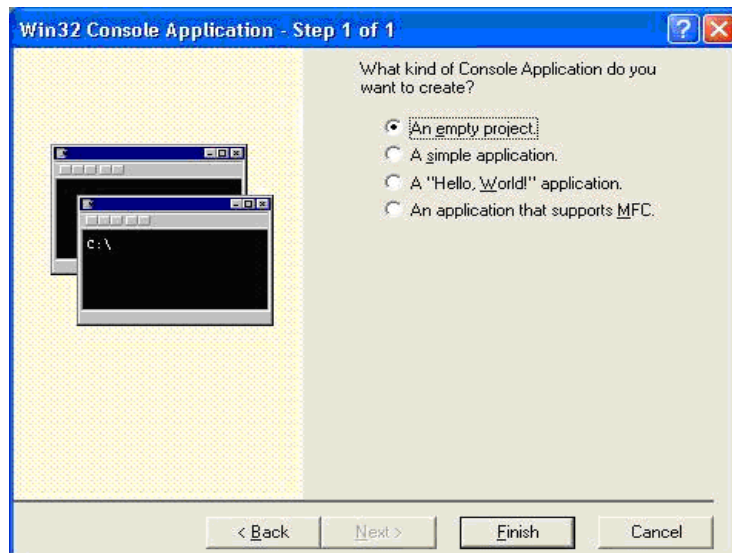


Рис. 2 — Второе диалоговое окно выбора проекта

Во втором диалоговом окне нажимаем Ok. После выполнения команды Finish появляется третье окно, которое показано на рис. 3.

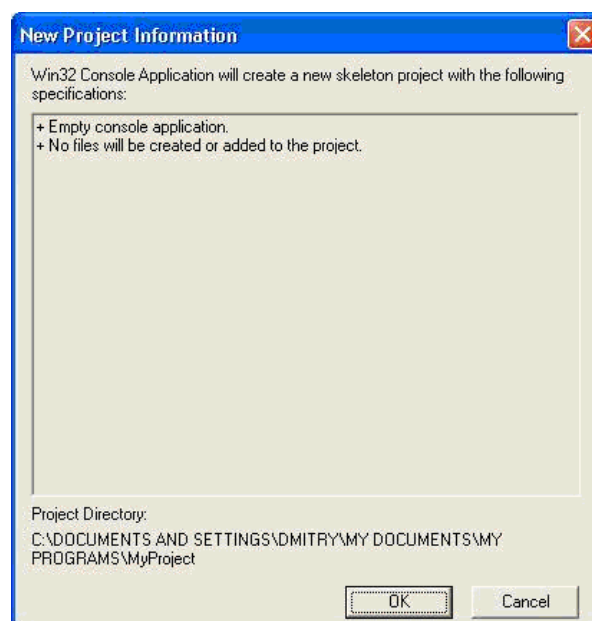


Рис. 3 — Третье диалоговое окно выбора проекта

Вот и все, наш проект готов, но в нем ничего нет. Теперь нам необходимо добавить файлы. Пока ограничимся всего одним файлом. В дальнейшем, когда необходимы будут несколько файлов или другие параметры проекта, мы вернемся к теме создания проектов и добавления файлов. А пока выполняем следующие действия. Выполнить команду: File — New (или нажать клавиши Ctrl+N). После чего появится диалоговое окно, аналогичное окну создания проекта. Вид этого окна приведен на рис.4. В появившемся диалоговом окне приведено множество типов файлов, с которыми может работать VC++, однако для проектов C++ , как вам известно, необходимо выбрать *.cpp и *.h.

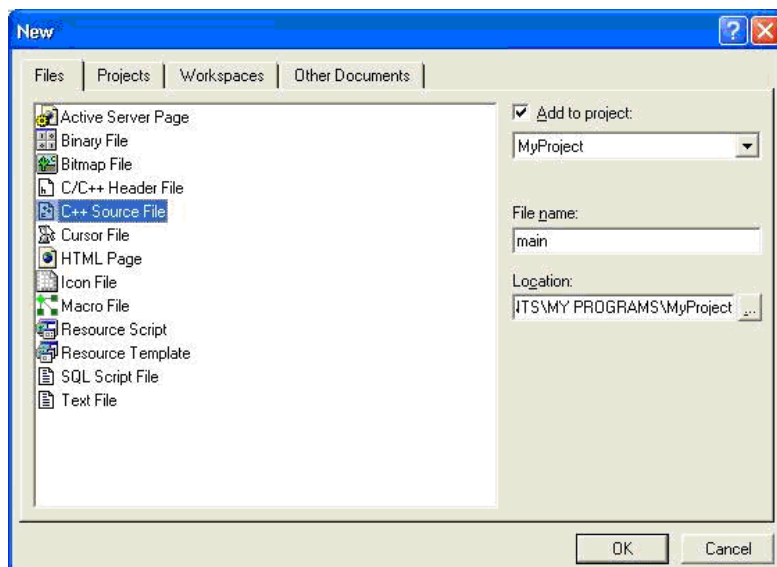


Рис. 4 — Выбор типа файла для проекта (в нашем случае C++)

Здесь выбираем тип файла C++ Source File, в поле File name вводим название нашего файла (в данном случае это будет main). Нажимаем кнопку Ok.

Теперь у нас проект готов. Давайте напишем первую программу.

Итак, в созданном нами файле пишем следующий код:

```
#include <iostream.h>
void main()
{
    cout << "Hello World" << endl;
```

}

Разбирать текст программы мы не будем, так как он прост (необходимо вывести на экран фразу «"Hello World"»). Мы же просто запустим программу. Для этих целей можно отдельно производить компиляцию, отдельно компоновку и после этого запускать программу, однако все эти действия можно выполнить одновременно. Для этого нажимаем клавиши Ctrl+F5 (есть и другие способы выполнения этого действия, но это самое простое, на мой взгляд). Появится окно сообщения, нажимаем Ok в этом окне. Если программа написана без синтаксических ошибок и вы выполнили все действия правильно, то программа будет откомпилирована, скомпонована и запущена на выполнение. Вы увидите результат выполнения вашей программы, аналогичный тому, что приведен на рис. 5.

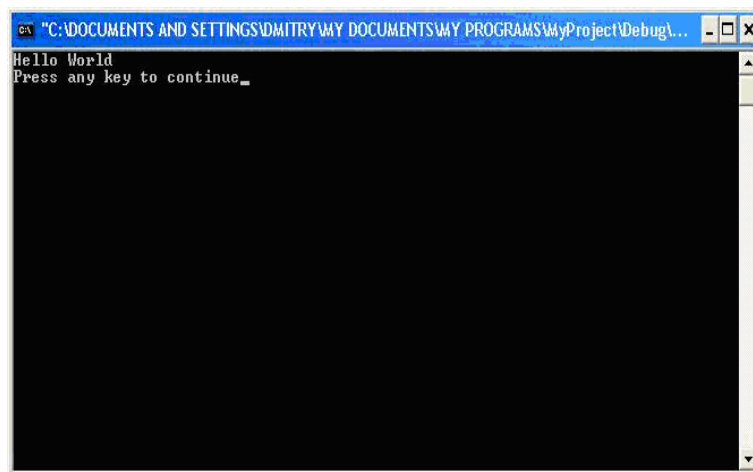


Рис. 5 — Результат выполнения первой программы

Вот и все, проект готов, написана первая программа. Теперь можете переходить к изучению всех остальных примеров.

1.9 Добавление файлов и классов в проект

На данном этапе мы рассмотрим добавление файлов в проект, а также добавление классов. В среде Visual C++ при добавлении классов при помощи встроенного «конструктора» для каждого класса создается свой файл. Мы не будем отходить от этого и тоже пока будем создавать для каждого класса свой файл. Но

это не стоит принимать как руководство к действию, так как в некоторых случаях это бесполезно или может отрицательно сказаться на всем проекте. Выше уже было сказано, что классы в проект можно добавлять посредством конструктора, который добавляет много разнообразной информации, которая на первых шагах может сильно запутать неопытного программиста. Поэтому пока мы будем добавлять классы вручную, но в последующих шагах рассмотрим способ добавления классов посредством конструктора.

Создадим свой класс, описание которого поместим в файл `MyClass.h`, а его реализацию поместим в файл `MyClass.cpp`.

Для выполнения этого откроем наш проект, который мы создали в шаге создания проекта. Для открытия проекта нужно выполнить следующее действие: запускаем среду Visual C++, в меню выбираем "File", а здесь, в свою очередь, выбираем пункт "Open Workspace". Обратите внимание, что выбираем этот пункт, а не просто функцию "Open", так как пункт "Open" предназначен для открытия других файлов, а не файла проекта. После того как Вы выполнили вышеописанное действие, появится диалоговое окно открытия проекта, см. рис. 6.

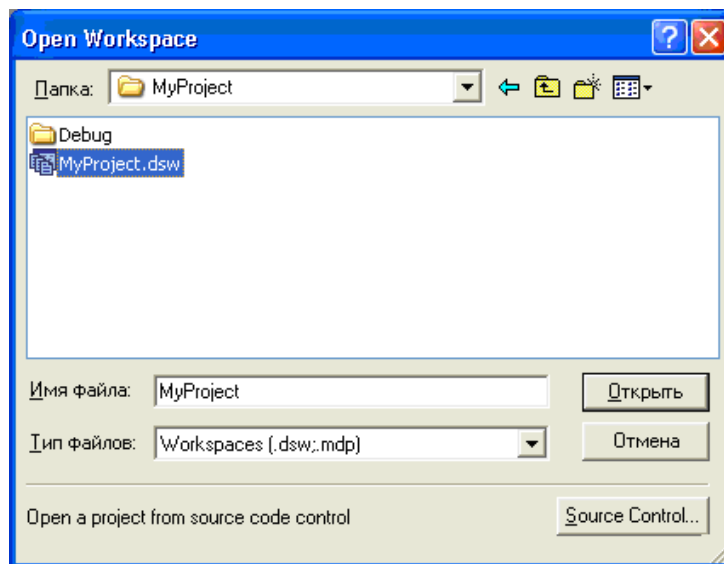


Рис. 6 — Диалоговое окно "Open Workspace"

Здесь мы "добираемся" до нашего файла проекта. В данном случае — это будет файл `MyProject.dsw`. Выбираем его и нажима-

ем "Открыть". Стоит отметить, что файлы проектов имеют расширение dsw.

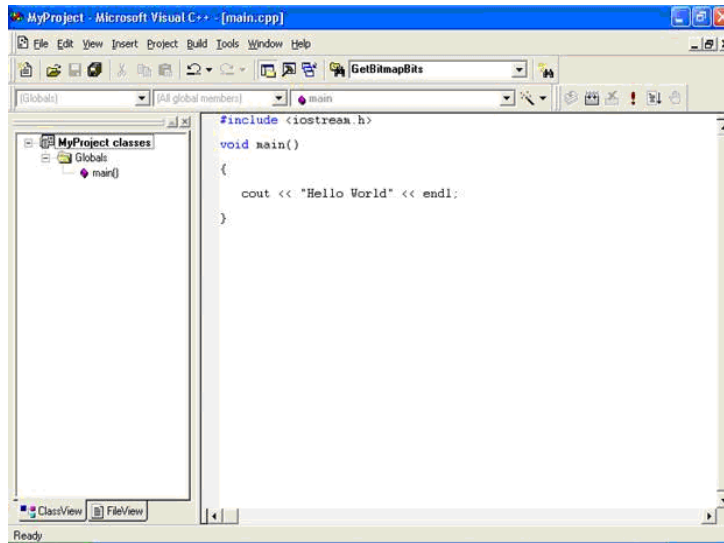


Рис. 7 — Результат открытия проекта

После открытия проекта вы должны увидеть следующий результат, как на рис. 7.

Немного рассмотрим окно среды Visual C++ 6.0. Так, в левой части экрана Вы можете увидеть менеджер проекта, в котором имеется несколько закладок (в данном случае 2, но в большинстве 3). Первая закладка ClassView. На ней отображаются классы, их методы и поля, а также глобальные функции. Вторая закладка, которую мы здесь наблюдаем, — это FileView, на которой отображаются все файлы нашего проекта (см. рис. 8).

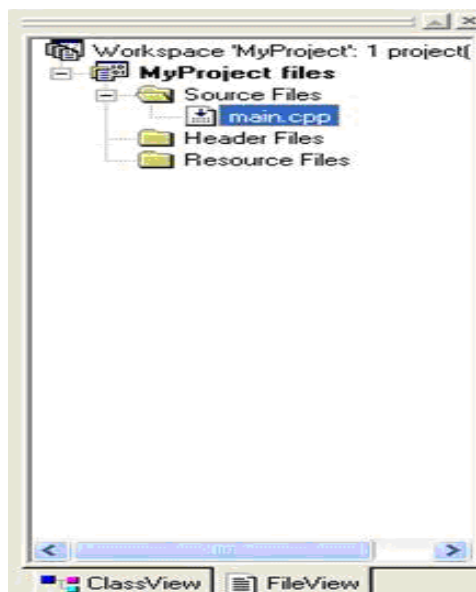


Рис. 8 — Менеджер проектов
(открыта опция "FileView")

Проект мы открыли, теперь перейдем к созданию класса. Для этого добавим еще два файла к нашему проекту (MyClass.h и MyClass.cpp). В этих целях выполним следующие действия. Выбираем пункт "New" в меню "File" (или, что на мой взгляд, проще, нажимаем сочетание клавиш Ctrl+N). После этого появится диалоговое окно, которое было рассмотрено в шаге создания проекта (там же мы добавляли файл main.cpp немного ранее). Здесь для добавления файла MyClass.cpp производим те же действия, только изменяем имя файла. После того, как файл *.cpp добавлен, добавим файл MyClass.h. Для этого выполняем аналогичные действия с той лишь разницей, что выбираем опцию выбора типа файла проекта "C/C++ Header File", а не "C++ Source File".

Теперь подключим наш файл (MyClass.h) к проекту. Для этого открываем файл main.cpp (не через Open, а через FileView) и в нем дописываем такую строку: #include "MyClass.h". Заметьте, что стандартные файлы мы описываем (равнозначно — вызываем) с помощью операторов C++ < >, а созданные пользователем — через " ". Теперь мы получим такой код в файле main.cpp:

```
#include <iostream.h>
#include "MyClass.h"
void main()
{
    cout << "Hello World" << endl;
}
```

Теперь немного преобразуем наш файл main.cpp и напомним некоторый текст в остальных файлах. Описывать действия программы не будем, так как мы здесь рассматриваем работу со средой VC++, а не объектное программирование.

После внесения изменений имеем:

```
//Файл main.cpp
#include "MyClass.h"
void main()
```

```
{
    CPrint pr("Hello World");
    pr.Show();
}
```

//Файл MyClass.h

```
class CPrint
```

```
{
char m_str[100];
```

```
public:
```

```
    CPrint(const char *pStr);
    void Show();
    ~CPrint();
};
```

//Файл MyClass.cpp

```
#include "MyClass.h"
```

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

```
#include <iostream.h>
```

```
CPrint::CPrint(const char *pStr)
```

```
{
    strcpy(m_str, pStr);
}
```

```
CPrint::~~CPrint()
```

```
{
cout << "Destroy object" << endl;
}
```

```
void CPrint::Show()
```

```
{
cout << m_str << endl;
}
```

После написания всего исходного текста собираем программу и запускаем на выполнение (как это делается, мы уже рассматривали).

Если все было сделано правильно и вы не допустили ни одной ошибки при написании исходных текстов, то получите результат, показанный на рис. 9.

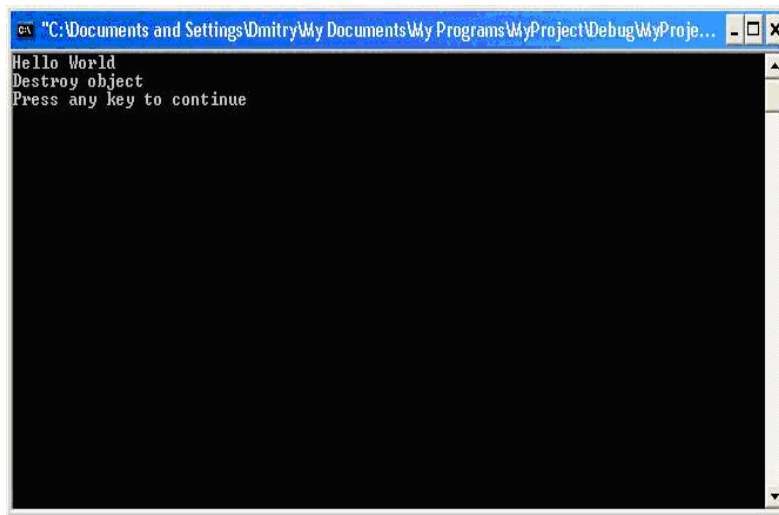


Рис. 9 — Результат выполнения исправленного проекта

Если же были допущены ошибки, то компилятор выдаст об этом сообщение. Как использовать эти сообщения для облегчения поиска ошибок, мы рассмотрим в другом шаге.

1.10 Создание классов посредством мастера

В этой статье мы рассмотрим создание классов при помощи мастера. Для этого мы создадим новый пустой консольный проект с именем WizardProject (как создавать проект, я надеюсь, Вы уже усвоили). После создания нового проекта добавьте в него файл `main.cpp`.



Рис.10 — Верхняя часть окна

менеджера проектов.

На рис. 10 мы видим часть закладки, на которой отображается наш проект. Для добавления нового класса к проекту необходимо выполнить следующие действия: установить указатель мыши на строку, отображающую название проекта (в нашем примере это WizardProject classes), щелкнуть правой кнопкой мыши и в появившемся меню выбрать пункт <New Class:>. После выполнения этого действия вы увидите диалоговое окно, в котором мы указываем название нового класса. Мастер автоматически генерирует названия h и cpp файлов, при желании мы можем изменить эти названия.

На рис. 11 приведен пример диалогового окна, о котором говорилось выше.

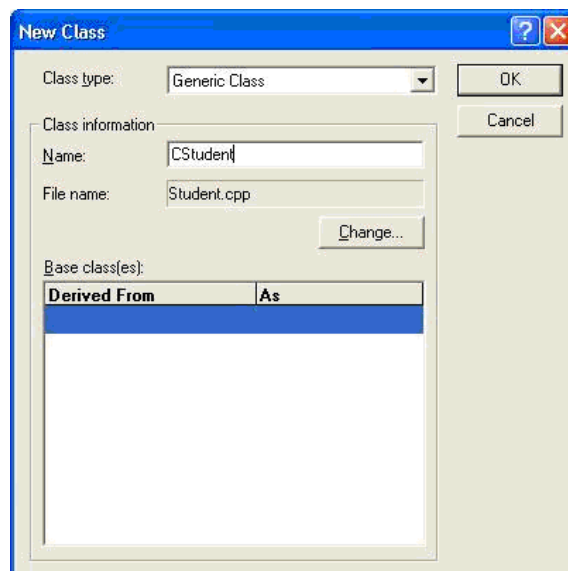


Рис.11 — Окно формирования данных о разрабатываемом классе

Здесь в поле <Name> мы вводим название нашего нового класса. Для нашего примера создадим класс CStudent. Поля и методы этого класса будем добавлять на другом шаге.

Теперь рассмотрим возможность изменения имен *.h и *.cpp файлов. Для изменения имен этих файлов нажмем кнопку <Change:>. После чего появится диалоговое окно, пример которого приведен на рис. 12.

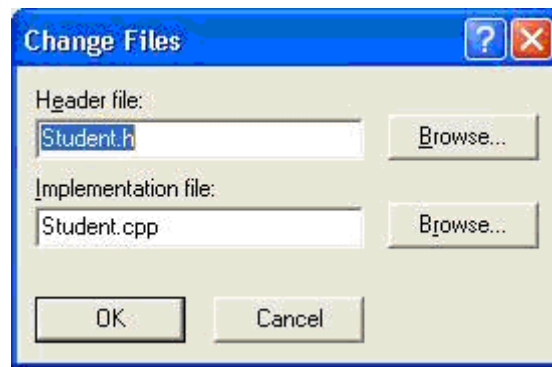


Рис.12 — Окно изменения имен файлов проекта

1.11 Добавление полей и методов мастером

В этой части мы рассмотрим добавление полей данных и методов к нашему проекту, созданному на прошлом шаге, посредством мастера. Для этого откроем проект WizardProject.

Добавим к нашему классу CStudent такие поля данных:

```
char m_szName[100] (фамилия),
int m_nAge (возраст),
char m_szGroup[100] (группа, в которой обучается студент).
```

Добавим методы:

```
void SetName(const char *pName) (метод, посредством которого будем за-
давать фамилию студента),
void SetAge(int nAge) (метод, посредством которого будем задавать возраст
студента),
void SetGroup(const char* pGroup) (метод для задания группы, в которой
обучается студент),
void Show() (метод, для отображения всех установленных данных).
```

Для добавления вышеописанных полей выполняем: на вкладке ClassView щелкаем правой кнопкой мыши по классу CStudent и выбираем пункт меню <Add Member Variable:>, как это показано на рис. 13. После выполнения этого действия появится диалоговое окно, аналогичное приведенному на рис. 14.

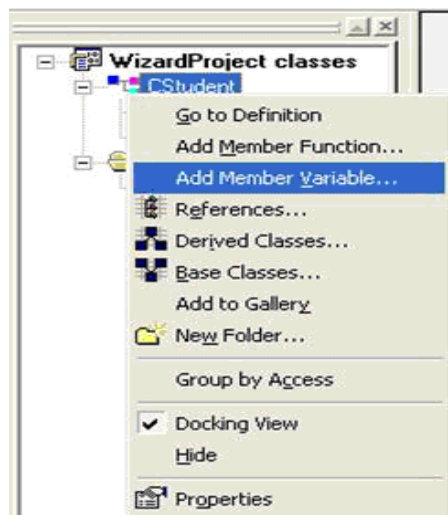


Рис.13 — Выбор опции добавления полей данных

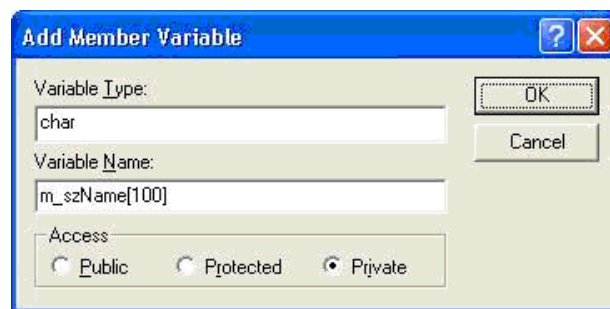


Рис.14 — Окно занесения полей данных

Здесь в поле <Variable Type> вводим тип поля, а в поле <Variable Name> указываем имя поля (но так как мы имеем дело с массивом/строкой, то указываем и размерность массива). В этом же окне указываем модификатор доступа (в данном случае private) и нажимаем <Ok>. Добавление остальных полей аналогично, за исключением поля возраста, там указывать размерность не нужно.

Практически также производится добавление методов класса. Немного рассмотрим этот процесс. Для добавления нового метода классу производим действия, аналогичные описанным выше, за исключением того, что в меню выбираем пункт <Add Member Function:>. Появится диалоговое окно, аналогичное окну добавления полей класса; пример окна приведен на рис. 15.

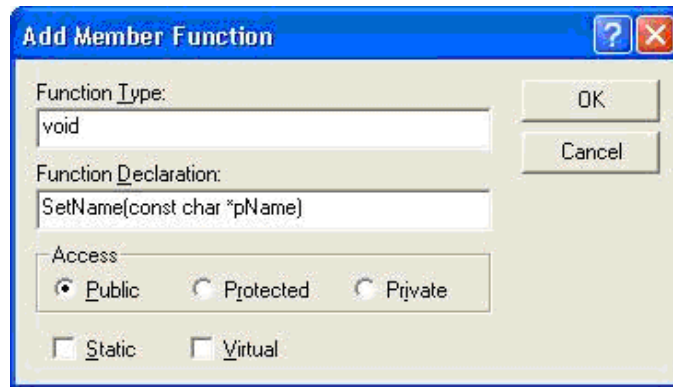


Рис. 15 — Окно добавления полей класса

Здесь в поле <Function Type:> указываем тип функции, в поле <Function Declaration:> объявляем метод, также здесь можем указать модификатор доступа и нажимаем Ok. Таким же образом добавляем остальные методы класса CStudent.

Теперь внесем некоторые изменения. После этого файлы должны выглядеть следующим образом:

```
//файл main.cpp
#include "CStudent.h"
void main()
{
    CStudent Student;
    Student.SetName("Ivanov");
    Student.SetAge(20);
    Student.SetGroup("925-15");
    Student.Show();
}
//файл CStudent.cpp
// CStudent.cpp: implementation of the CStudent class.
#include "CStudent.h"
#include <string.h>
#include <iostream.h>
// Construction/Destruction
CStudent::CStudent() : m_nAge(0)
{
    memset((void*)m_szName, 32, 100);
    memset((void*)m_szGroup, 32, 100);
}
CStudent::~~CStudent()
```

```
{  
}  
void CStudent::SetName(const char *pName)  
{  
    strcpy(m_szName, pName);  
}  
  
void CStudent::SetAge(int nAge)  
{  
    m_nAge = nAge;  
}  
void CStudent::SetGroup(const char *pGroup)  
{  
    strcpy(m_szGroup, pGroup);  
}  
void CStudent::Show()  
{  
    cout << "Name:\t" << m_szName << endl  
        << "Age:\t" << m_nAge << endl  
        << "Group:\t" << m_szGroup << endl;  
}
```

Собираем программу и запускаем на выполнение. Если все сделано безошибочно, то Вы увидите такой результат, который показан на рис. 16.

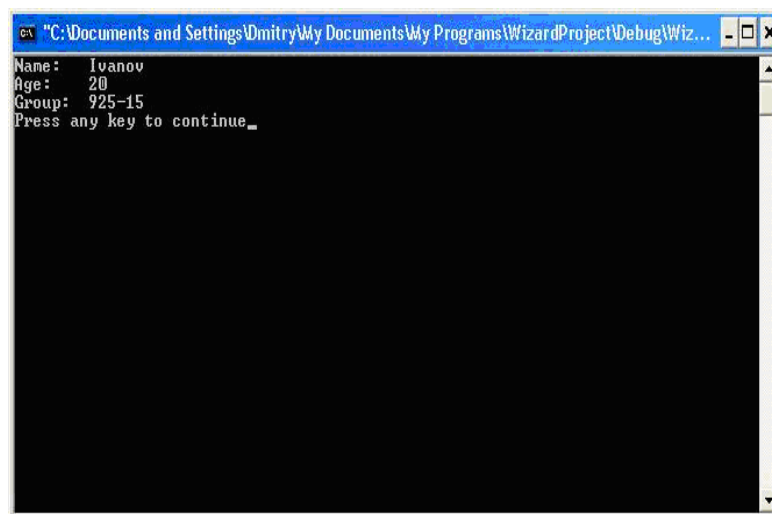


Рис. 16 — Окно результата запуска программы

Отметим, что, несмотря на солидный возраст данного компилятора и среды разработки, он пользуется заметным успехом благодаря эффективности и надежности разрабатываемых в нем программ. Среда разработки проста и понятна начинающим программистам.

2 СРЕДА РАЗРАБОТКИ VISUAL C++ 2005

На замену компилятора VC++ 6.0 фирма Microsoft разработана новую линейку компиляторов, которая теперь стала нумероваться не цифрами, а годом выпуска. Начиная с компилятора VC++2005, структура оконного интерфейса среды разработки мало меняется и данного в пособии материала достаточно, чтобы понять основной принцип работы последующих компиляторов (VC++2008, VC++2010 и VC++2012).

2.1 Пользовательский интерфейс

Внешний вид пользовательского интерфейса среды программирования приведен на рис. 17.

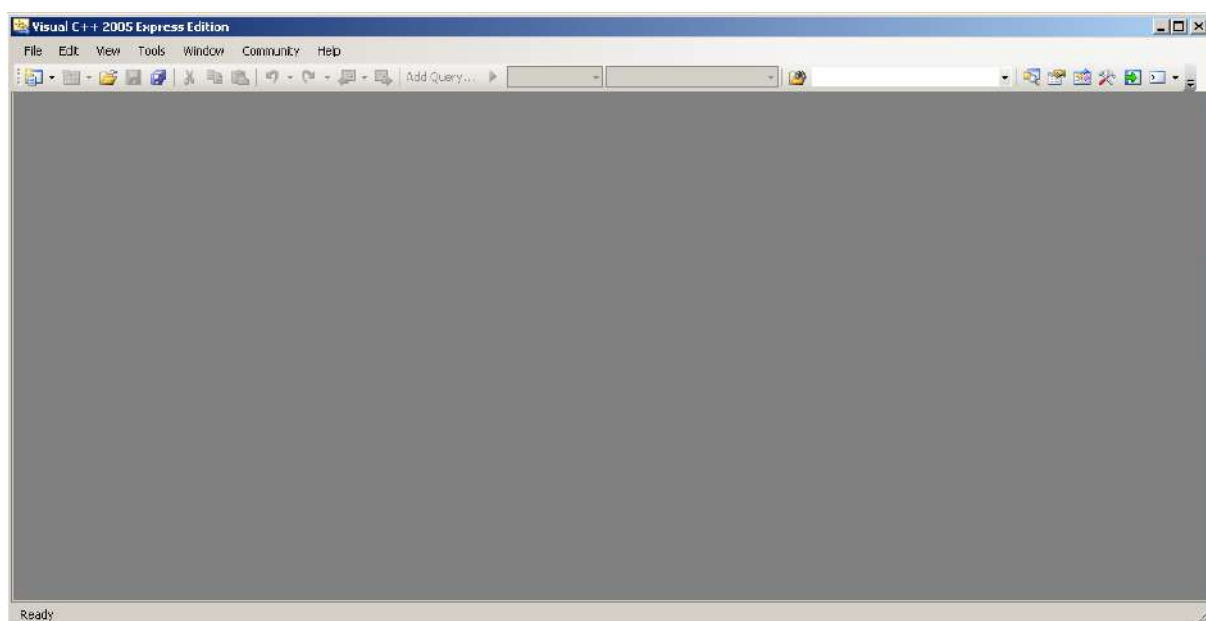


Рис.17 — Внешний вид окна рабочего пространства среды

Для графического отображения объектов рабочей области (см. рис. 18) используется список с древовидным отображением, корневыми узлами которого являются проекты. Содержимое этих проектов можно представить тремя способами, каждому из которых соответствует вкладка в нижней левой части рабочей области:

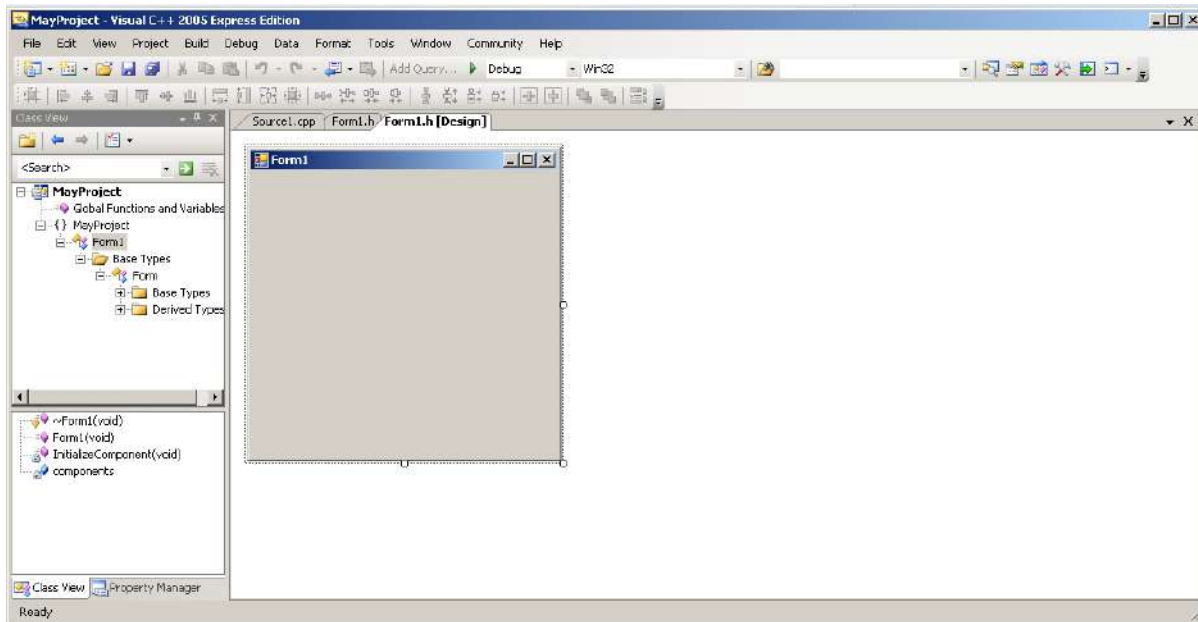


Рис.18 — Графическое отображение объектов на вкладке **ClassView**

ClassView — представляет программу в объектно-ориентированном виде, отображая классы C++, их методы и члены-данные (левая верхняя панель, см. рис. 18). Двойной щелчок такого объекта вызовет переход к его объявлению или реализации.

Project Manager — отображает ресурсы, сгруппированные по категориям (см. рис. 19).

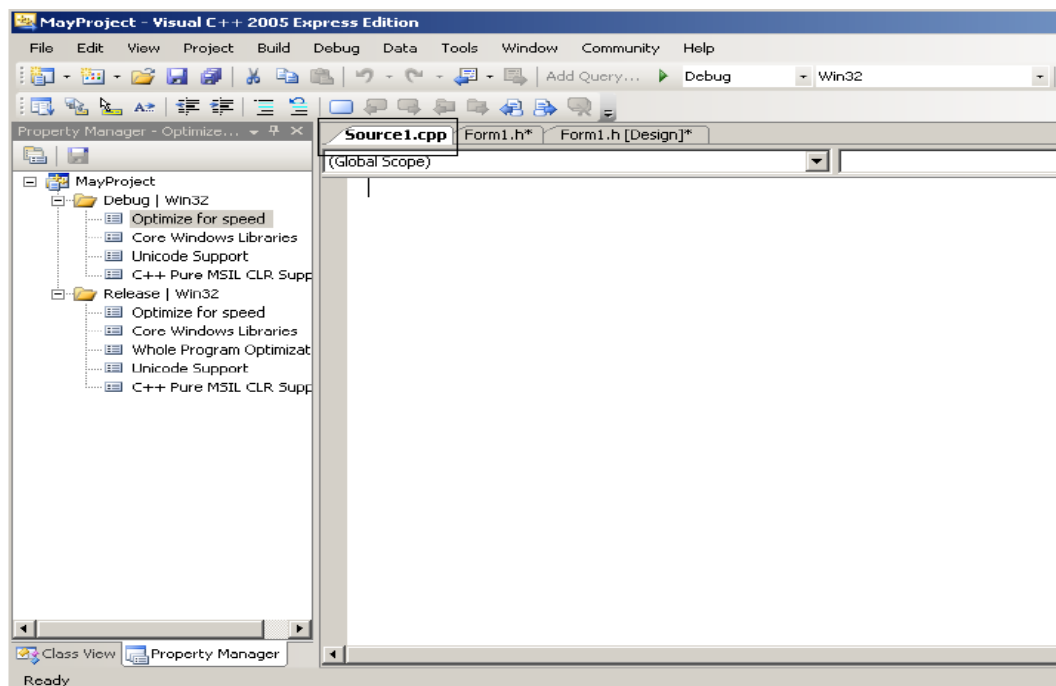


Рис. 19 — Графическое отображение объектов на вкладке **Project Manager**

Show All Files — показывает все файлы проекта, которые можно редактировать (см. рис. 20).

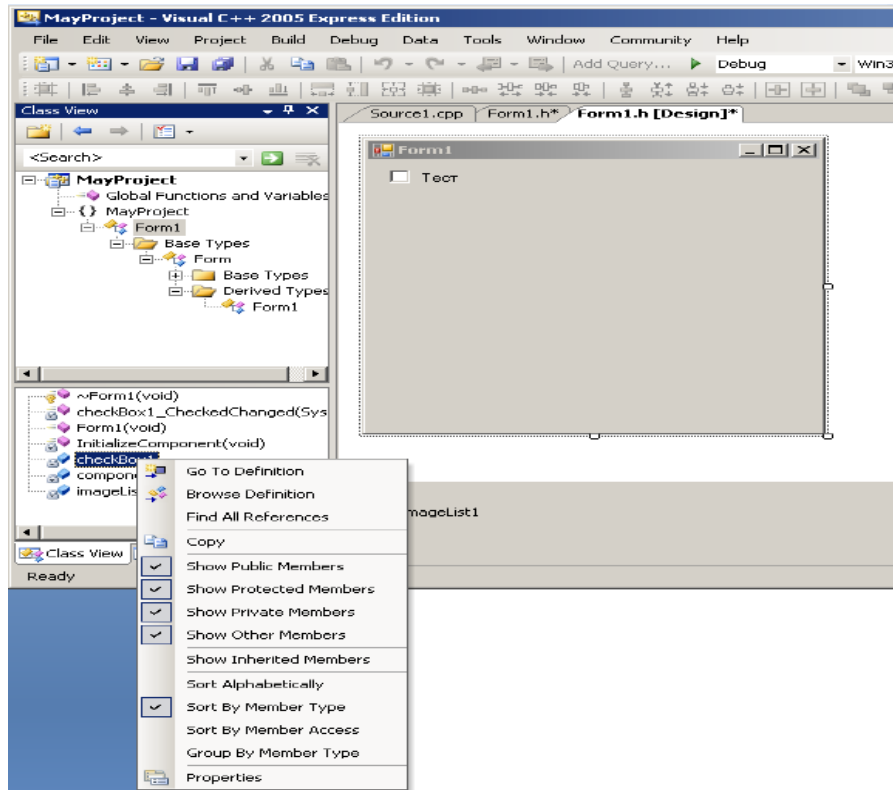


Рис. 20 — Графическое отображение объектов на вкладке **Show All Files**

При щелчке правой кнопкой мыши на объекте появится контекстное меню, пункты которого связаны с этим объектом.

2.2 Меню и панели инструментов

Среда разработки Visual C++ Express Editor обладает набором меню, позволяющим управлять файлами и рабочими областями проектов, настраивать саму среду, а также обращаться к справочной системе, программе управления исходным кодом и внешним инструментальным средствам. Почти каждому меню соответствует панель инструментов, в которой команду можно выбрать одним щелчком кнопки мыши. Панели инструментов допустимо настраивать — добавлять и удалять кнопки, скрывать и отображать сами панели инструментов. Поэтому Вы можете

сконфигурировать среду по своему вкусу, упростив доступ к часто используемым вами пунктам меню.

По умолчанию в окне среды Visual C++ отображены две панели инструментов. Панель **Standard** (содержит команды, часто используемые при работе с файлами), **Layout** (представляет инструмент для работы с объектами на форме).

При щелчке правой кнопки мыши на поле расположения панелей (например, в пустой правой верхней части панели инструментов) появится полный список возможных панелей, позволяющий скрывать и отображать панели (см. рис. 21).

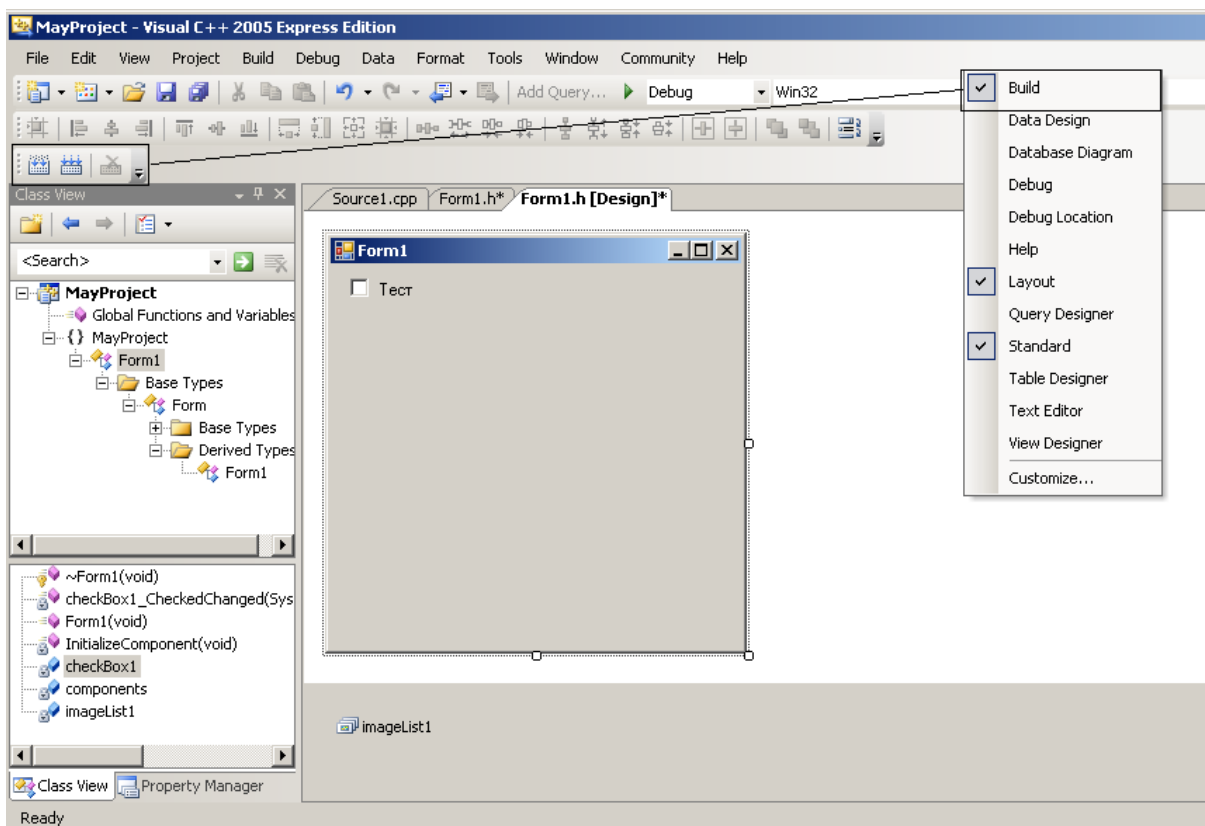


Рис. 21 — Выбор содержания панелей среды разработки

Для настройки пользовательского интерфейса необходимо проделать следующие действия:

1. В меню **Tools** выберите пункт **Customize**. Появится диалоговое окно **Customize** (см. рис. 22).
2. Перейдите на вкладку **Commands**.
3. В списке **Category** выберите пункт **View**.

4. Щелкните значок **Full Screen** (третий слева в первом ряду). Обратите внимание на описание команды, которое появляется под раскрывшимся списком **Category**.

5. Перетащите значок **Full Screen** на панель инструментов **Standart**.

6. Закройте окно **Customize**.

7. Откройте файл с исходным кодом программы, дважды щелкнув его в окне **FullView**.

8. Испытайте новую кнопку — щелкните ее, чтобы перейти в полноэкранный режим. Чтобы вернуться к традиционному виду, необходимо снова щелкнуть эту кнопку.

Чтобы удалить кнопку с панели инструментов, достаточно щелкнуть ее правой кнопкой мыши и выбрать из контекстного меню пункт **Delete**.

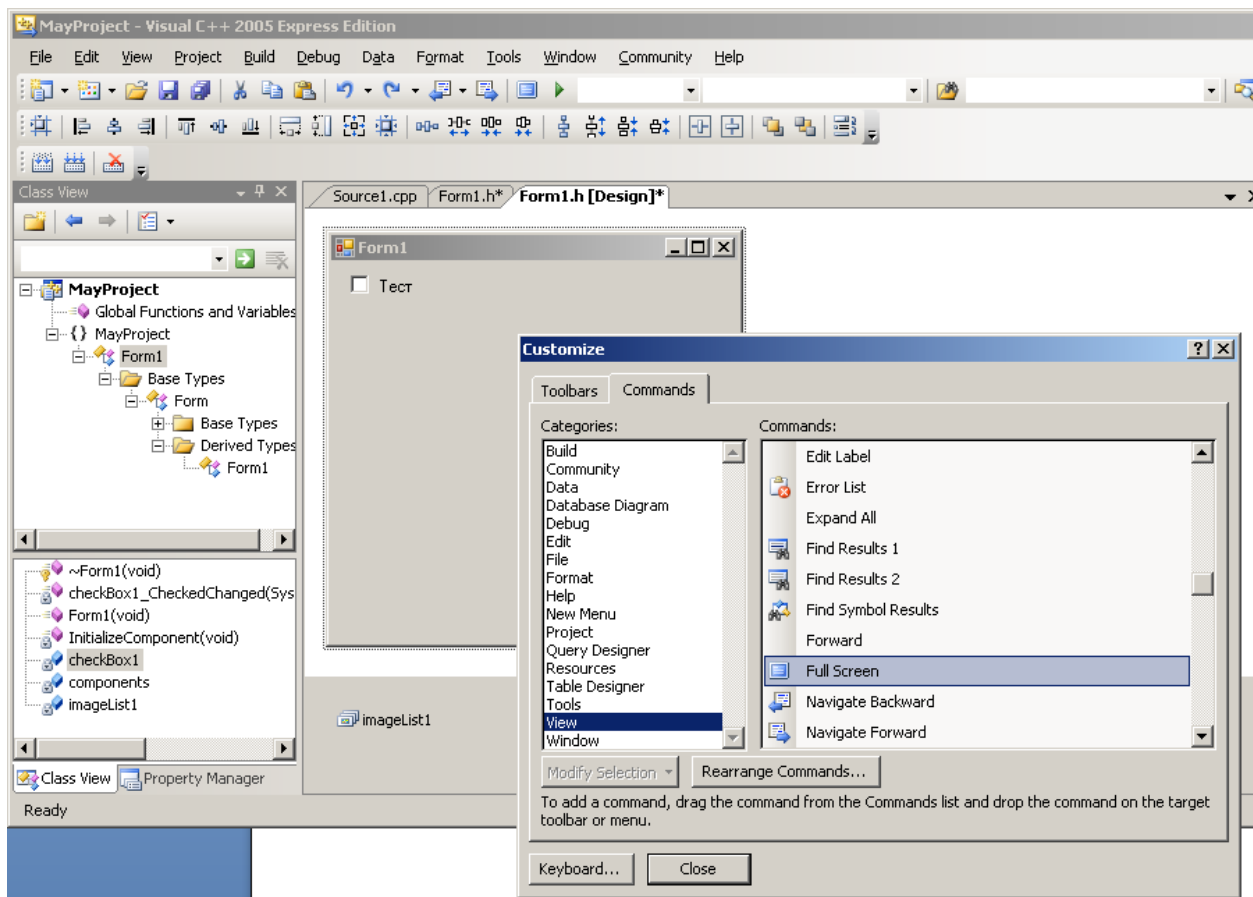


Рис. 22 — Выбор опции **Customize** и команд входящих в ее состав

2.3 Настройка параметров среды

Выбрав в меню Tools пункт Options, можно вызвать диалоговое окно Options (см. рис. 23). В этом окне настраиваются параметры вашей рабочей среды, от которых зависят вид и возможности встроенных в нее инструментальных средств. Данные параметры применяются ко всем проектам и их конфигурациям.

В диалоговом окне можно настроить:

- 1) редактор кода — начиная с параметров окна и сохранения и заканчивая параметрами таких функций, как автозаполнение кода и цветовое выделение синтаксических элементов. Можно эмулировать некоторые версии редакторов;
- 2) отображение информации отладчиком;
- 3) параметры встроенной программы управления исходным кодом;
- 4) пути к исполняемым файлам, файлам с исходным кодом, включаемым файлам и библиотекам.

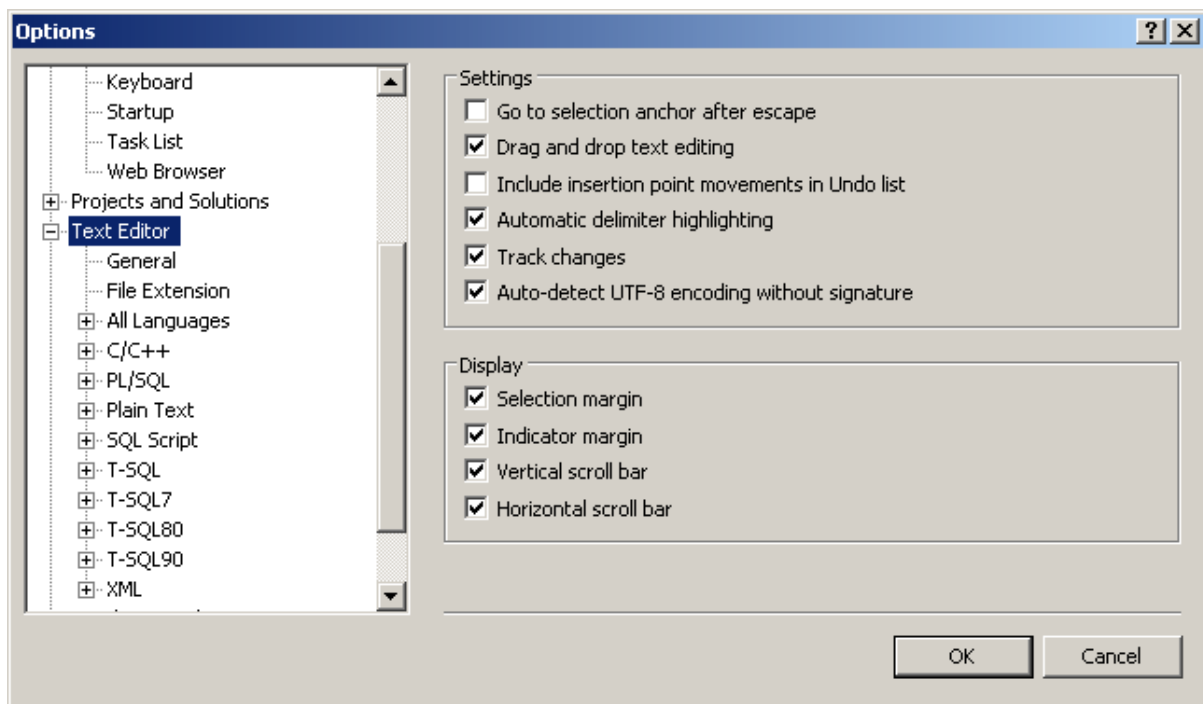


Рис. 23 — Работа с окном параметров Options

2.4 Параметры конфигурации проекта

При работе над проектом в некоторых случаях возникает необходимость изменения параметров конфигурации проекта. Для этого вызовите окно **Property**, выбрав в меню **Project** пункт **Properties**. В этом окне можно настроить параметры проекта, в том числе и параметры компиляторов и компоновщика Visual C++.

Важно отметить, что изменения параметров на любой вкладке влияют только на текущий проект и версию сборки, указанные в раскрывающемся списке **Configuration**. Для каждого типа сборки можно задать свой набор параметров. Если требуется установить одинаковые параметры для всех версий сборки, в списке **Configuration** выберите **All Configurations**.

Не менее важно при настройке проекта проверить, какую его конфигурацию вы изменяете.

Приведем описание содержания некоторых вкладок диалогового окна **Project Settings** (см. рис. 24).

- **General** — здесь можно указать, компоновать ли программу с использованием статических библиотек MFC (если Вы решили изменить способ компоновки, выбранной Вами в окне мастера **Wizard**). Также можно задать каталоги, в которые будут помещаться конечные файлы.

- **Debugging** — Здесь находятся параметры, передаваемые программе при ее запуске в отладчике. Кроме того, можно использовать переадресацию ввода/вывода, аналогичную применяемой в командной строке.

- **C/C++** — здесь вы определяете параметры компилятора, функциональные возможности языка, соглашения о вызове, параметры, связанные с типом процессора, оптимизации кода, константами препроцессора и т. п.

- **Linker** — на этой вкладке выбирают дополнительные библиотеки, которые следует скомпилировать вместе с Вашей программой.

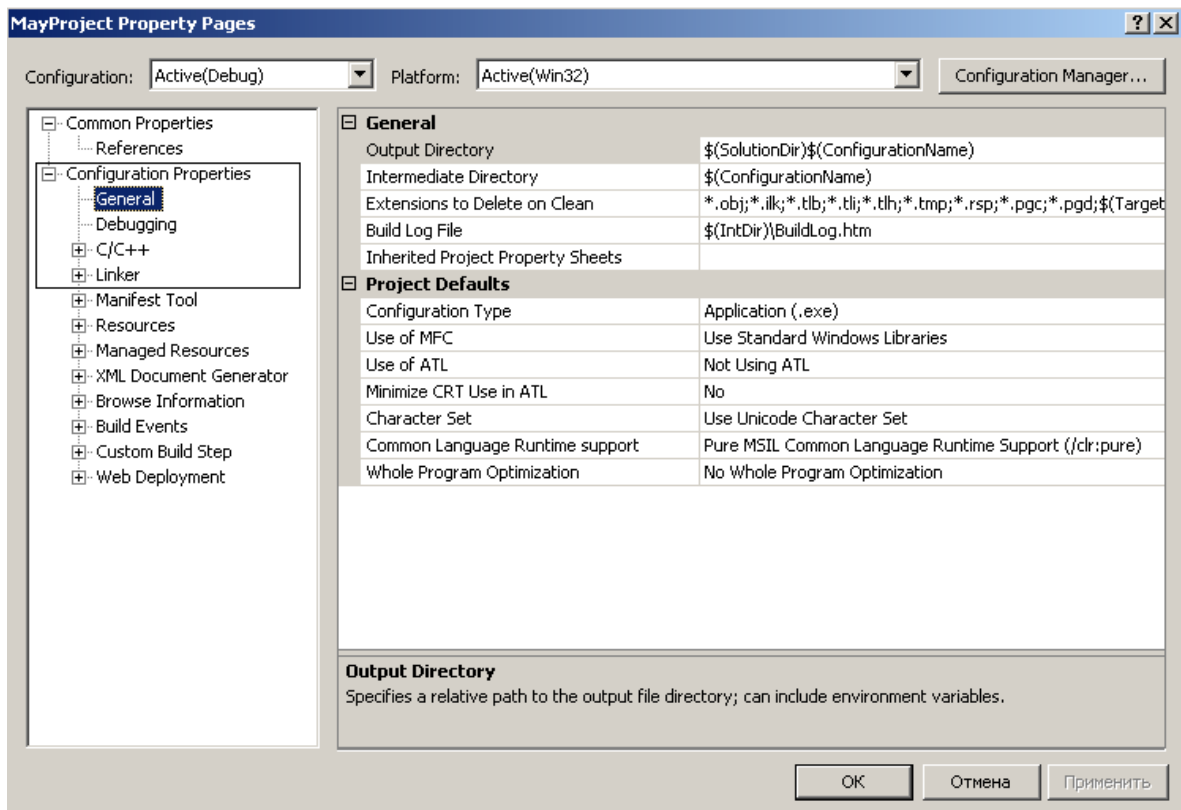


Рис. 24 — Параметры конфигурации проекта

2.5 Типы мастеров проекта

В среде Visual C++ можно строить различные типы проектов. Такие проекты после их создания можно компилировать и запускать на исполнение. Фирма Microsoft разработала специальный инструмент, облегчающий и ускоряющий создание проектов в среде Visual C++.

Рассмотрим некоторые типы проектов, которые можно создавать при помощи различных средств (мастеров проектов) Microsoft Visual C++:

MFC AppWizard (exe) — при помощи этого мастера приложений можно разработать проект Windows-приложения, имеющего однодокументный, многодокументный или диалоговый интерфейс. Однодокументное приложение может предоставлять возможность пользователю в любой момент времени работать только с одним документом в окне. Многодокументное приложение, напротив, может одновременно представлять несколько документов, каждый в собственном окне. Пользовательский интерфейс диалогового приложения представляет собой единственное диалоговое окно.

MFC AppWizard (dll) — этот мастер приложений позволяет создать структуру DLL, основанную на MFC. При помощи него можно определить характеристики будущей DLL.

AppWizard ATL COM — это средство позволяет создать элемент управления ActiveX или сервер автоматизации, используя новую библиотеку шаблонов ActiveX (ActiveX Template Library — ATL). Опции этого мастера дают возможность выбрать активный сервер (DLL) или исполняемый внешний сервер (exe-файл).

Custom AppWizard — при помощи этого средства можно создать пользовательские мастера AppWizard. Пользовательский мастер может базироваться на стандартных мастерах для приложений MFC или DLL, а также на существующих проектах или содержать только определяемые разработчиком шаги.

DevStudio Add-in Wizard — мастер дополнений позволяет создавать дополнения к Visual Studio. Библиотека DLL расширений может поддерживать панели инструментов и реагировать на события Visual Studio.

MFC ActiveX ControlWizard — мастер элементов управления реализует процесс создания проекта, содержащего один или несколько элементов управления ActiveX, основанных на элементах управления MFC.

Win32 Application — этот мастер позволяет создать проект обычного Window-приложения. Проект создается незаполненным, файлы с исходным кодом в него следует добавлять вручную.

Win32 Console Application — мастер создания проекта консольного приложения. Консольное приложение — это программа, которая выполняется из командной строки окна DOS или Windows и не имеет графического интерфейса (окон). Проект консольного приложения создается пустым, предполагая добавление файлов исходного текста в него вручную.

Win32 Dynamic-Link Library — создание пустого проекта динамически подключаемой библиотеки. Установки компилятора и компоновщика будут настроены на создание DLL. Исходные файлы следует добавлять вручную.

Win32 Static Library — это средство создает пустой проект, предназначенный для генерации статической (объектной) библиотеки. Файлы с исходным кодом в него следует добавлять вручную.

2.6 Выбор типа проекта

Для создания нового проекта выберите в меню File =>New=>Project, на экране появится форма мастера проекта; в зависимости от решаемой вами задачи выберите тип проекта, дайте название вашему проекту, далее нажмите кнопку Ok. Мастер подготовит необходимые формы и сгенерирует код — так называемую оболочку для дальнейшей разработки вашего проекта.

Создание проекта. В этом окне выбираем тип проекта Win32 Consol Application (см. рис. 25), так как пока будем работать с консольным проектом. В поле name вводим название нашего проекта, в нашем случае это MyProject. Теперь нажимаем кнопку Ok. После этого появятся еще два диалоговых окна. В первом ничего не меняем, а просто нажимаем кнопку Finish, т.е. создадим пустой проект.

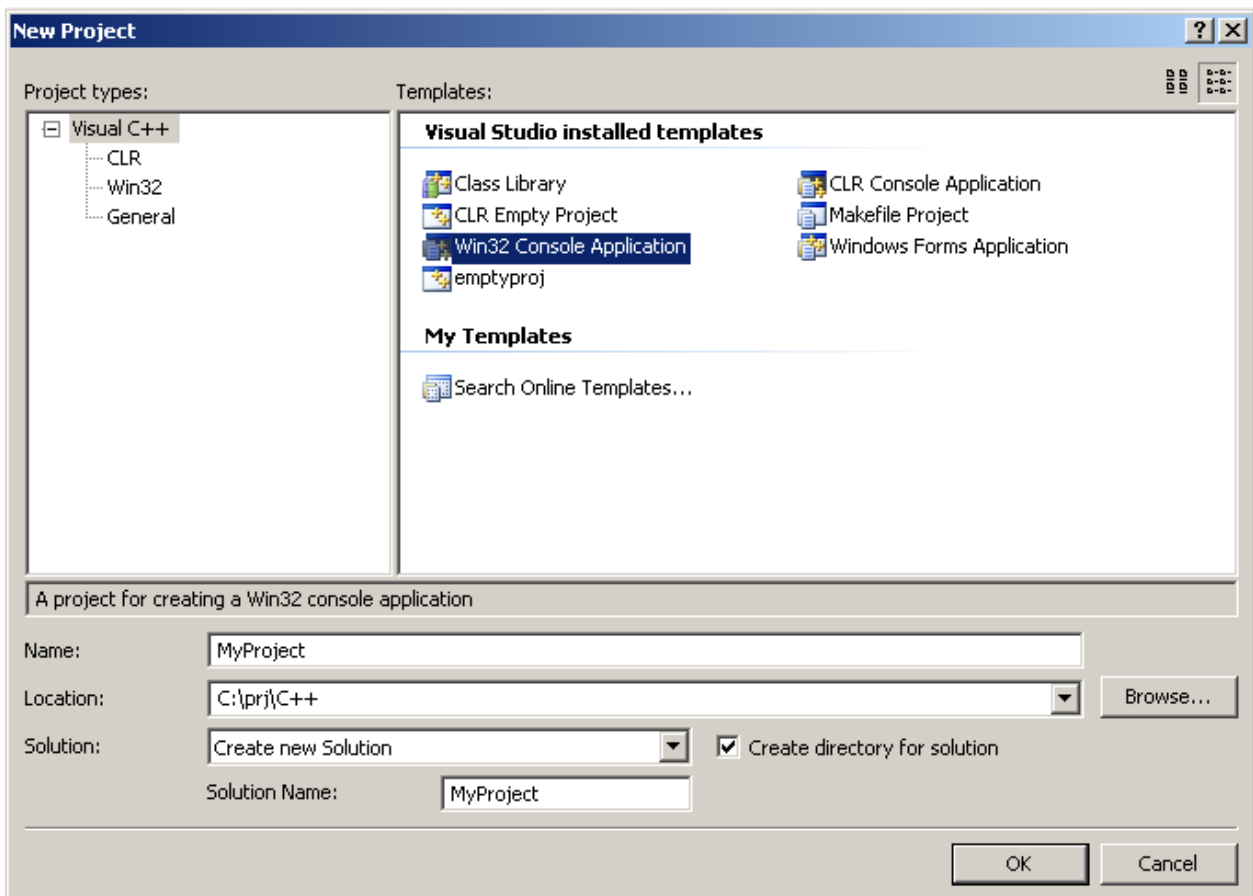


Рис. 25 — Окно выбора типа проекта

2.7 Добавление файлов в проект

Создадим свой класс, описание которого поместим в файл `MyClass.h`, а его реализацию поместим в файл `MyClass.cpp`.

Для выполнения этого откроем наш проект, который мы создали в шаге создания проекта (см. рис. 26). Для открытия проекта нужно выполнить следующее действие: запускаем среду Visual C++, в меню выбираем "File", а здесь, в свою очередь, выбираем пункт "Open=>Project". Обратите внимание, что выбираем этот пункт, а не просто функцию "Open". Так как пункт "Open" предназначен для открытия других файлов, а не файла проекта программы.

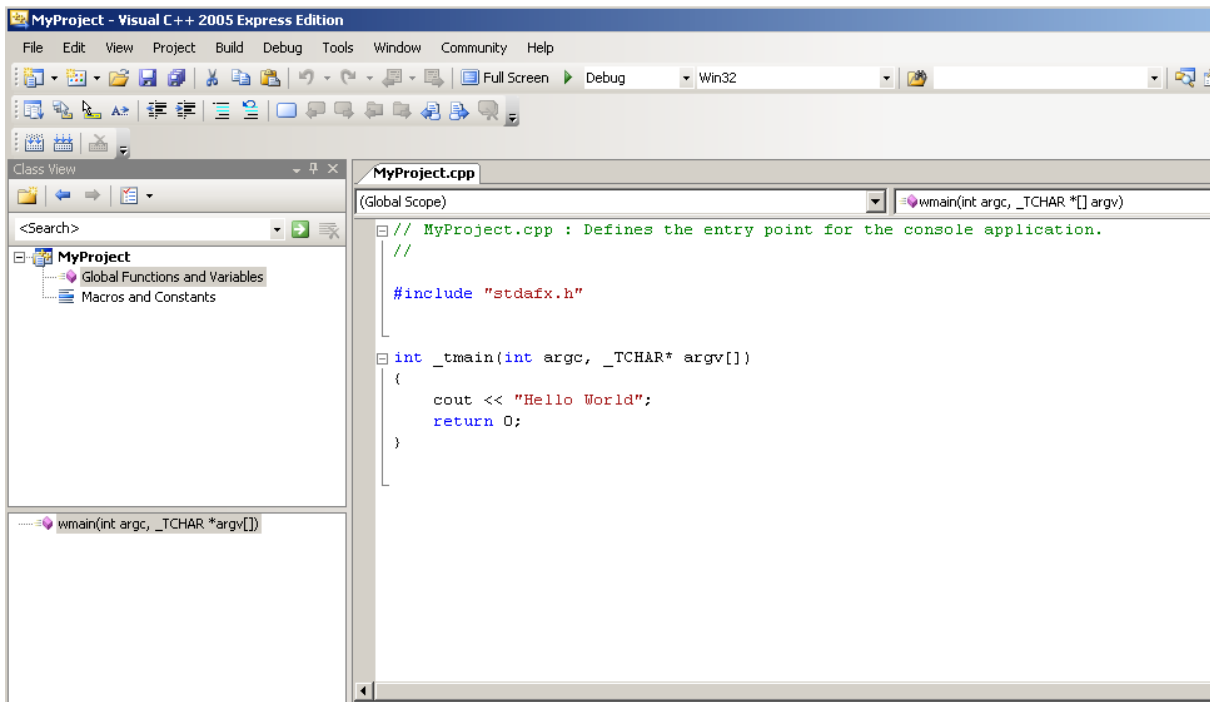


Рис. 26 — Элементы первой программы в среде программирования Visual C++

Теперь подключим наш файл (`MyClass.h`) к проекту. Для этого открываем файл `main.cpp` (не через `Open`, а через `FileView`) и в нем дописываем такую строку: `#include "MyClass.h"` (см. рис. 27).

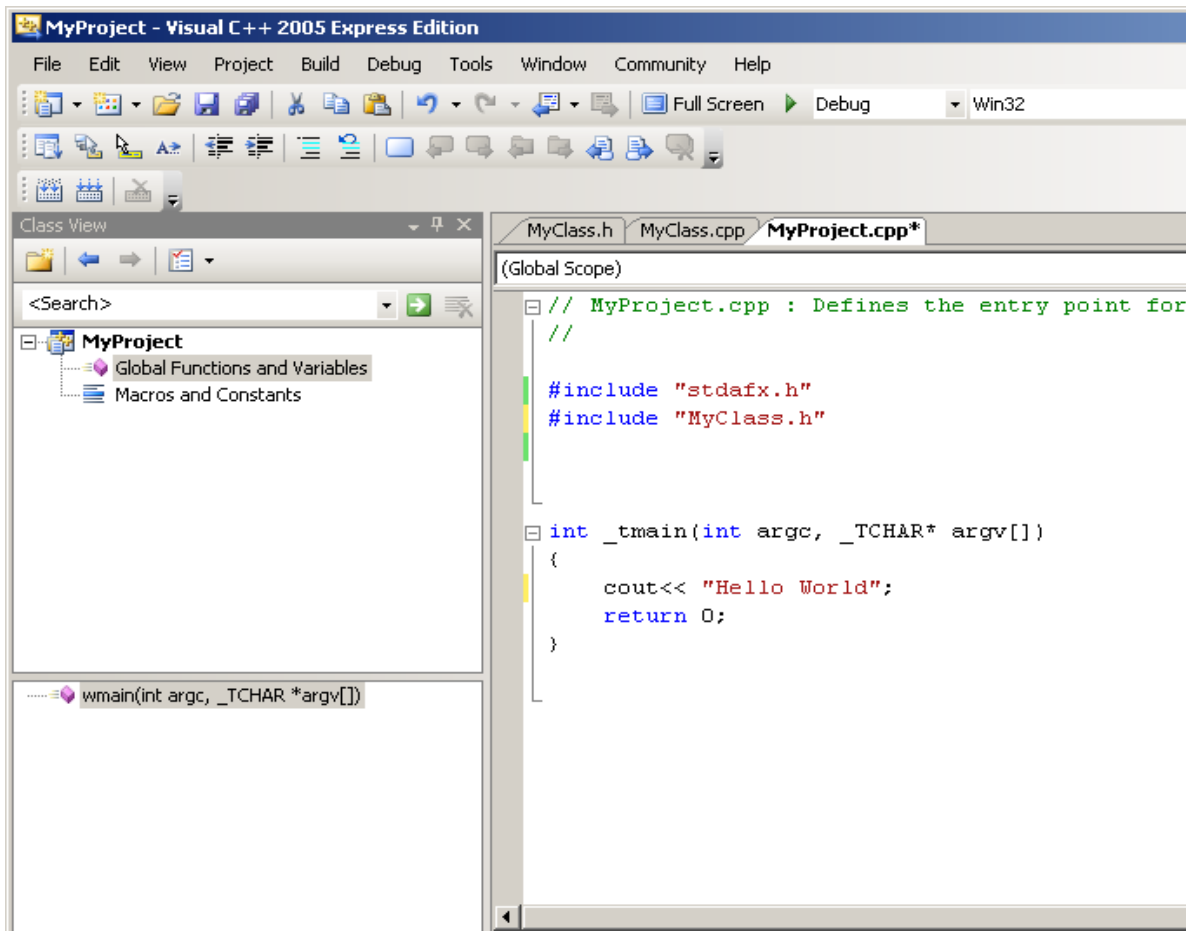


Рис. 27 — Добавление подключаемых файлов в проект

2.8 Создание классов по средством мастера

Рассмотрим некоторые действия для создания классов при помощи мастера. Для этого мы создадим новый пустой консольный проект с именем WizardProject (см. рис. 28). После создания нового проекта добавьте в него файл `main.cpp`.

Для добавления нового класса к проекту необходимо выполнить следующие действия: установить указатель мыши на строку, отображающую название проекта (в нашем примере это WizardProject classes (см. рис. 29)), щелкнуть правой кнопкой мыши и в появившемся меню выбрать пункт `Add => Class`. После выполнения этого действия Вы увидите диалоговое окно, в котором мы указываем название нового класса. Мастер автоматически генерирует названия `h` и `cpp` файлов, при желании мы можем изменить эти названия.

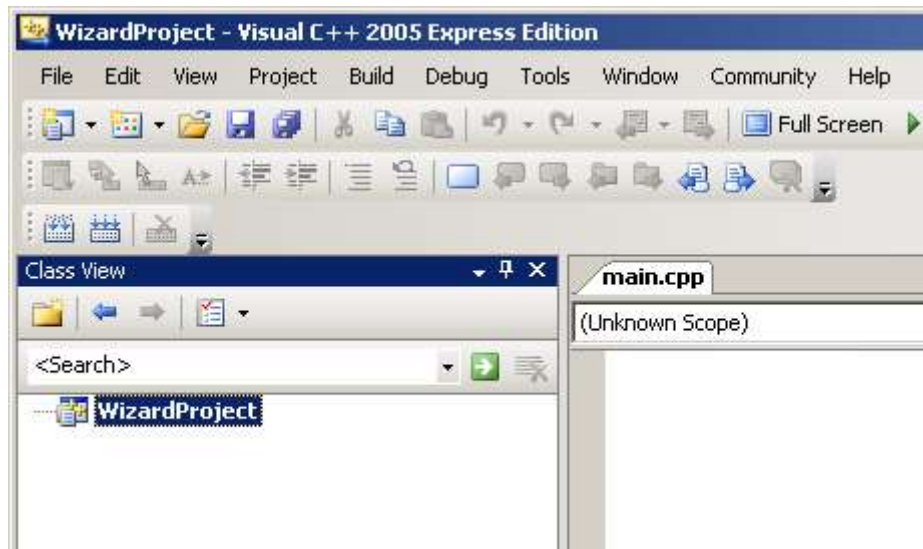


Рис. 28 — Окно интерфейса создания нового консольного проекта с именем WizardPro

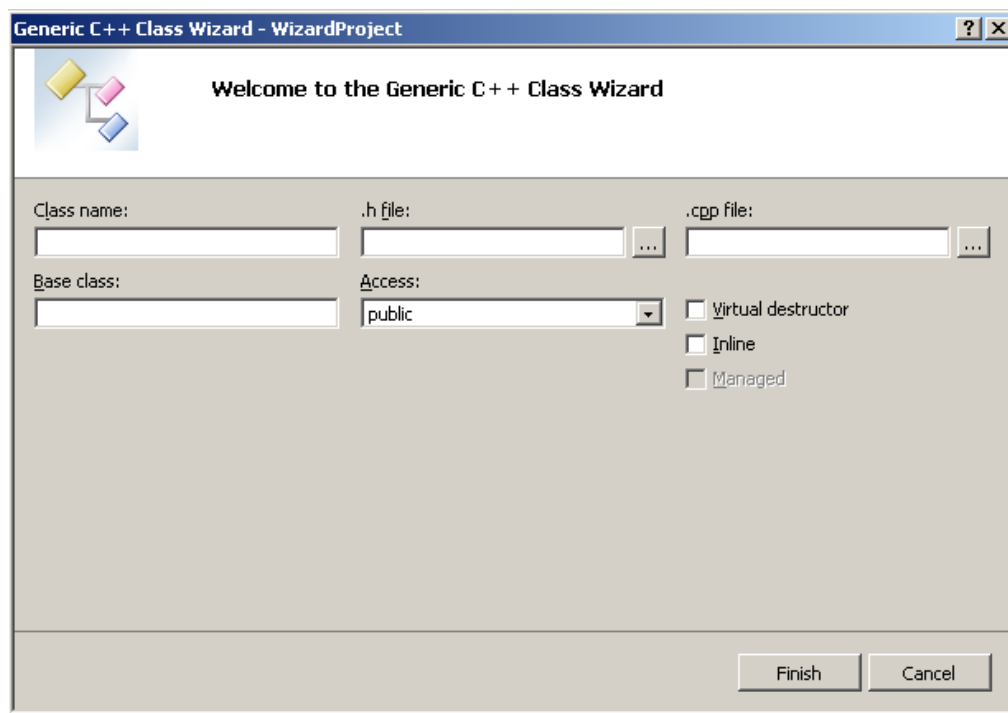


Рис. 29 — Окно формирования данных о разрабатываемом классе

Здесь в поле <Class Name> мы вводим название нашего нового класса. Для нашего примера создадим класс CStudent (см. рис. 30). Далее нажимаем кнопку Finish. Поля и методы этого класса будем добавлять на другом шаге (см. рис. 31).

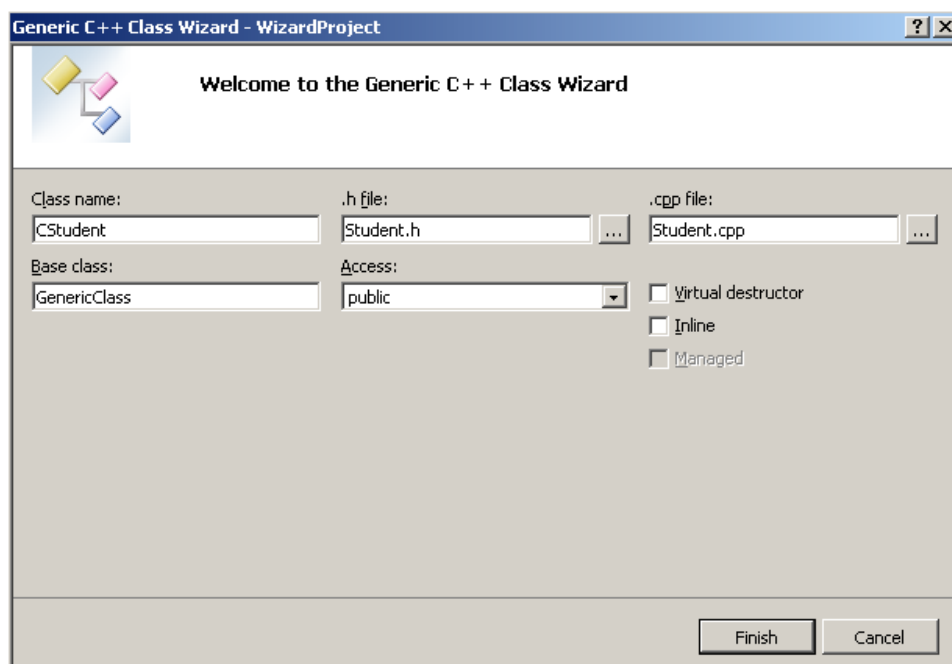


Рис. 30 — Заполнение элементов окна создания файлов проекта

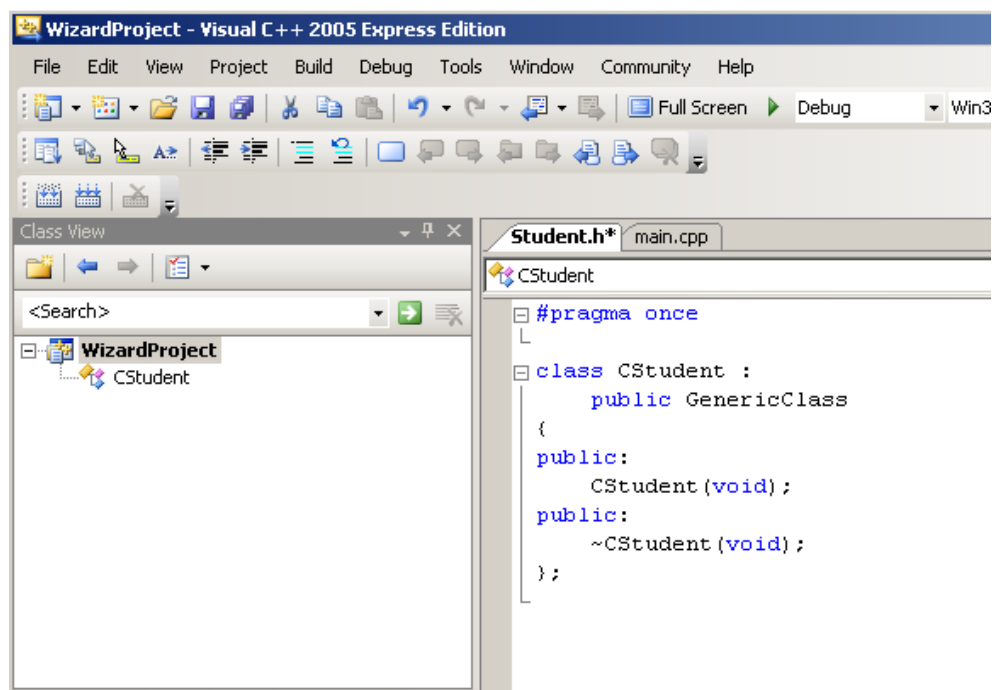


Рис. 31 — Окно созданного нашего класса и добавленного в проект

На рис. 31 показано заполнение поля программы Student.h элементами программы.

2.9 Пример программы вывода текста на экран

```
#include <iostream>
void main()
{
    /*Эта программа выводит вопрос
    как вас зовут ?*/
    char name[20];
    std::cout<<"What is your name:\n";
    std::cin >>name;
    std::cout<<"Hello:"<<name;
}
```

Результата работы программы:

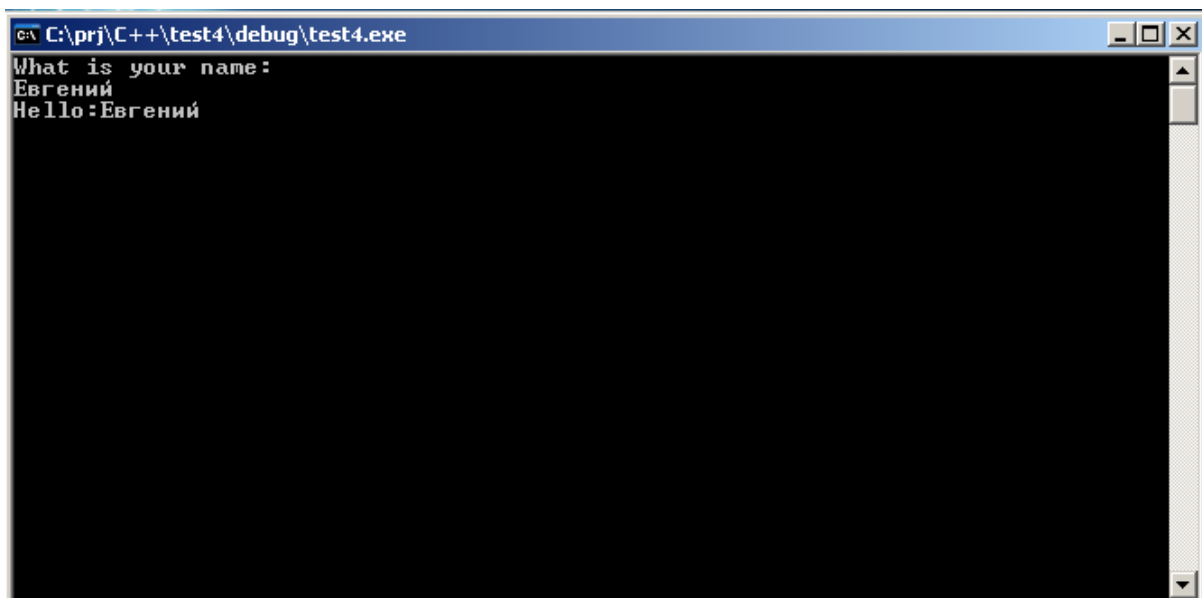


Рис. 32 — Результат работы программы

На рис. 32 показан результат запуска программы, который определяется выводом вопроса об имени пользователя, чтении имени пользователя и последующего отображения этого имени.

3 ЛАБОРАТОРНЫЕ РАБОТЫ ПО ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

Выбор варианта лабораторных работ осуществляется по общим правилам с использованием следующей формулы:

$$V = (N * K) \text{ div } 100,$$

где V — искомый номер варианта,

N — общее количество вариантов,

div — целочисленное деление,

при $V = 0$ выбирается максимальный вариант,

K — код варианта.

Варианты заданий представлены в табл. 3.1, в которой дан минимальный набор параметров, необходимый для выполнения лабораторных работ. В представленных ниже двух лабораторных работах дан пример создания объекта с определенным набором данных. Студент вправе увеличивать число параметров объекта и определять им необходимый для этого тип, но не уменьшать их относительно табл. 3.1.

Таблица 3.1 — Варианты заданий описания членов — данных пользовательских классов

1. СТУДЕНТ имя — char* курс — int пол — int(bool)	2. ИЗДЕЛИЕ имя — char* шифр — char* количество — int	3. АДРЕС имя — char* улица — char* номер дома — int
4. ЦЕХ имя — char* начальник — char* количество работающих — int	5. СТРАНА имя — char* форма правления — char* площадь — float	6. СЛУЖАЩИЙ имя — char* возраст — int рабочий стаж — int
7. БИБЛИОТЕКА имя — char* автор — char* стоимость — float	8. ТОВАР имя — char* количество — int стоимость — float	9. ПЕРСОНА имя — char* возраст — int пол — int(bool)

Окончание табл. 3.1

10. ЖИВОТНОЕ имя — char* класс — char* средний вес — int	11. КАДРЫ имя — char* номер цеха — int разряд — int	12. ЭКЗАМЕН имя студента — char* дата — int оценка — int
13. КВИТАНЦИЯ номер — int дата — int сумма — float	14. АВТОМОБИЛЬ марка — char* мощность — int стоимость — float	15. КОРАБЛЬ имя — char* водоизмещение — int тип — char*
16. КНИГА номер по списку — int автор — char сумма — float	17. СОТОВЫЙ марка — char номер — int сумма — float	18. САМОЛЕТ рейс — int дата — int сумма — float
19. ПРИНТЕР марка — char номер — int сумма — float	20. ЛИНЕЙКА длина — int ширина — int сумма — float	21. ДОГОВОР номер — int руководитель — char сумма — float
22. КНОПКА диаметр — int тип — char длина жала — float	23. МЫШЬ марка — char число кнопок — int сумма — float	24. ЦВЕТОК тип — char дата — int сумма — float
25. СТУЛ число ножек — int число спинок — int сумма — float	26. КАРТА тип — char масштаб — int полушарие — char	27. СТЕКЛО толщина — int цвет — int марка — char

3.1 Работа № 1

Тема: *Классы. Открытые и закрытые уровни доступа. Конструкторы. Инициализация данных объекта. Определение методов. Создание объекта в памяти. Стандартные потоки ввода-вывода.*

1. В среде программирования на C++ создайте консольный проект с именем LAB1 в каталоге LAB1.

2. В проекте создайте файлы main.h (заголовочный файл) и main.cpp (файл исходного кода).

3. В файле main.h определите с помощью ключевого слова **class** объект Person.

Данные объекта:

Номер человека (целый тип).

ФИО (символьный массив).

Пол (логический тип: 0 — муж., 1 — жен.).

Возраст (вещественный тип).

Пусть данные имеют закрытый уровень доступа (**private**).

4. Опишите конструктор объекта, аргументы которого будут инициализировать все данные объекта.

5. Опишите конструктор объекта по умолчанию (без аргументов), проинициализировав все данные.

6. Опишите в объекте функцию void Print() с открытым уровнем доступа (**public**), которая будет выводить данные на экран.

7. Откройте файл main.cpp. С помощью директивы **#include** включите в файл main.cpp заголовочные файлы <stdlib.h>, <string.h>, <iostream.h>, а также ваш заголовочный файл "Main.h".

8. Ниже определите конструктор объекта, инициализирующий все данные объекта значениями аргументов. В теле конструктора используйте функцию strcpy(стр1, стр2) для копирования строки имени человека (ФИО).

9. Затем определите функцию void Person::Print(). В теле функции для вывода данных используйте стандартный поток вывода **cout** << значение1 << значение2 << ... << endl;.

10. Ниже напишите главную функцию программы **int main()**. Внутри ее создайте объект **Person**, указав все значения данных объекта. Выведите данные объекта на экран, вызвав функцию **Print**.

11. Затем создайте динамический объект **Person** с помощью обычного конструктора и оператора **new**. Выведите данные объекта на экран. Удалите динамический объект из памяти с помощью оператора **delete**.

12. Напишите функцию ввода данных в объект с клавиатуры **void Person::Input()**. В теле функции для ввода данных используйте стандартный поток ввода **cin >> значение1 >> значение2 >> ...;**

13. Затем в теле функции **main** создайте объект **Person** с помощью конструктора по умолчанию и введите данные в объект с клавиатуры, вызвав функцию **Input**. Выведите данные объекта на экран.

3.2 Работа № 2

Тема: *Создание динамического массива объектов. Деструктор объекта. Два типа полиморфизма: принудительное приведение типа, перегрузка функций и перегрузка операторов (унарных и бинарных).*

1. В среде программирования на C++ создайте консольный проект с именем **LAB2** в каталоге **LAB2**.

2. Переименуйте файл **main.h** из предыдущей лабораторной в **person.h**. Создайте файл **person.cpp** и включите в проект эти два файла. Переместите конструктор и функции объекта **Person** из **main.cpp** в файл **person.cpp**. Таким образом, файл **person.h** содержит описание объекта **Person**, а файл **person.cpp** — реализацию объекта **Person**.

3. Включите в проект файл **main.cpp** и очистите тело функции **main()**.

4. Определим объект **Group**, который будет содержать динамический массив объектов **Person**. Создайте два файла **group.h** и **group.cpp** и включите их в проект.

5. В файле **group.h** определите с помощью ключевого слова **class** объект **Group**.

Данные объекта:

размер массива (целый тип)
указатель на массив (тип `Person*`).

Пусть данные имеют закрытый уровень доступа (***private***).

6. Опишите конструктор объекта с одним аргументом — размер массива (целый тип) и деструктор объекта.

7. Откройте файл *group.cpp*. С помощью директивы ***#include*** включите необходимые заголовочные файлы.

8. Определите конструктор объекта *Group*. В теле конструктора проинициализируйте данные объекта, т.е. проинициализируйте размер массива значением аргумента конструктора и выделите динамическую память под массив с помощью строки кода:

указатель на массив = ***new*** *Person*[*размер массива*].

9. В деструкторе объекта освободите память, занимаемую массивом, с помощью строки кода

delete [*указатель на массив*].

Таким образом, мы создали объект *Group*, который содержит массив объектов *Person*.

10. Определим открытые (***public***) методы для объекта *Group*. Напишите функцию ***void Group::Print()***, которая выводит в цикле ***for*** все записи массива на экран. В теле цикла примените ранее написанную функцию *Print()* для объекта *Person*.

11. Напишите функцию ***int Group::Size()***, которая возвращает размер массива.

12. Напишите функции ***void PutPerson(int i, Person& man);*** и ***Person& GetPerson(int i);***, первая функция заносит объект *man* типа *Person* в *i*-й элемент массива, вторая функция возвращает объект типа *Person* из *i*-го элемента массива.

13. Заполните массив данными и затем выведите их на экран. Для этого в теле функции ***int main()*** сначала определите массив имен, которые будут заноситься в поле ФИО объекта *Person*, например,

```
char names[5][25] = {"A", "B", "C", "D", "E"};.
```

Затем создайте объект с именем *group* типа *Group* размером пять записей, т.е. *Group group(5);*.

14. Ниже с помощью цикла **for** заполните массив данными. Для этого в теле цикла создайте объект *Person*, проинициализировав все его данные, и с помощью функции *PutPerson* занесите объект в массив.

15. Выведите массив на экран с помощью строки кода: *group.Print();*. Получилось? Если да, то вы научились создавать динамический массив объектов, определять функции работы с таким массивом и выводить его на экран.

16. В этой части лабораторной работы изучим первые два типа полиморфизма — это а) принудительное приведение типа; б) перегрузка функций и операторов.

17. Напишите функцию приведения типа. Для этого с помощью ключевого слова **operator** напишите функцию объекта *Person*, которая преобразует тип *Person* в **double**. Пусть функция возвращает возраст человека, например,

```
Person::operator double() { return this->Age; }.
```

Что означает ключевое слово **this**?

18. Проверьте функцию преобразования типа. В функции *int main()* далее определите переменную **double** и присвойте ей объект *Person*, например:

```
double age = group.GetPerson(2);.
```

То есть совершается неявное преобразование из типа *Person* в тип **double** при обращении к объекту. Выведите значение переменной на экран.

19. Перегруженные функции имеют одинаковое название, но разный возвращаемый тип или/и разный список аргументов. Определим в объекте *Group* две функции с одинаковым именем, например **double Age();** и **double Age(int limit);**. Первая функция пусть возвращает средний возраст группы людей, а вторая функ-

ция пусть возвращает средний возраст людей в группе, возраст которых не больше некоторого граничного значения *limit*. Функции отличаются списком аргументов.

20. Проверьте работу перегруженных функций, отобразив на экране подсчитанные два значения среднего возраста.

21. Перегрузите оператор индексирования. Если раньше, чтобы обратиться к элементу массива, нам необходимо было вызывать функцию *GetPerson*, то, определив оператор индексирования, мы будем использовать только квадратные скобки. Сравните две строки кода:

```
Person man = group.GetPerson(2);
Person man = group[2];
```

В объекте *Group* с помощью ключевого слова ***operator*** определите оператор индексирования, например:

```
Person& Group::operator[](int i).
```

В теле оператора напишите код, возвращающий *i*-ый элемент массива, т.е. объект *Person*.

22. Выведите на экран с помощью оператора индексирования любой один элемент массива *group*, например третий.

23. Перегрузим бинарный оператор, например оператор сложения (+), для объекта *Person*. Пусть оператор сложения будет возвращать суммарный возраст двух человек. Опишем в объекте *Person* данный оператор как дружественную функцию с помощью ключевого слова ***friend***, например:

```
friend double operator+(Person& p1, Person& p2);
```

Эта строка кода означает, что оператор сложения не принадлежит объекту, но ему доступны все закрытые данные и методы объекта.

В файле *person.cpp* определите оператор сложения, например:

```
double operator + (Person& p1, Person& p2)
{ return (p1.Age + p2.Age); }
```

Здесь мы напрямую обращаемся к закрытому полю *Age* объекта *Person*.

24. Проверим работу оператора с помощью следующих строк кода:

```
double sum = group[1] + group[3];  
cout << sum << endl;
```

4 ПРАВИЛА ОФОРМЛЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

а) Лабораторная работа состоит в выполнении задания, описания материала в формате MS WORD и подготовке файлов проекта (файлы *.exe, *.cpp, *.h). Выполнение задания может быть выполнено в любой среде программирования C++, но с обязательным ее указанием в файле отчета (дело в том, что существуют некоторые отличия программирования в средах, например Visual Studio и Borland).

б) Лабораторная работа обязательно должна содержать:

- краткое изложение теоретического материала по каждому из рассматриваемых разделов в лабораторной (например, указатель, наследование, др.);
- комментарии по тексту программы, описывающие основные особенности объектно-ориентированного программирования на C++;
- дополнения (создания нового класса, применение наследования, полиморфных и дружественных функций и др.) и изменения, приводящие к усложнению варианта лабораторных работ.

в) Окончательно сформированное задание содержит файл описания *.word и файлы проекта *.exe, *.cpp, *.h.