Тема 1

Основные понятия

Элементы языка программирования

1. Алфавит

Программа на языке программирования записывается с помощью символов, образующих алфавит языка. Алфавит языка C++ включает:

- большие латинские буквы от A до Z
- малые латинские буквы от a до z
- цифры от 0 до 9
- знаки препинания: , . ; : ! ?
- скобки: ( ) [ ] { }
- знаки математических операций: + -* / <> =
- специальные символы: \ (обратная наклонная черта)
- ~ (волна или тильда)
- & (амперсант)
- # (решетка или диез)
- ’ (апостроф или одиночная кавычка)
- ” (двойная кавычка)
- ^ (стрелка)
- % (процент)
- _ (знак подчеркивания)

Элементы языка программирования

2. Лексемы

Лексема – это фрагмент программы, имеющий самостоятельное значение. Различают следующие виды лексем:

1) Ключевое слово – это лексема, имеющая некоторое предопределенное значение в языке. Ключевые слова нельзя использовать для каких-либо других целей.

2) Идентификатор – это последовательность букв, символа «_» и цифр, начинающаяся с буквы или символа «_». Идентификаторы используются как имена объекта программы .

3) Числовые константы – задают в программе конкретные целые или дробные числа, записываемые по обычным правилам, например, 123, 32.1, 3.21e2.

4) Строковые константы – это последовательности произвольных символов, заключенные в двойные кавычки, например, ”Строка символов”, ”String of characters”.

5) Символьные константы представляют в программе одиночные символы, при записи заключаются в одиночные апострофы, например, ’a’, ’A’, ’0’, ’1’.

6) Термином **оператор** в языке C++ обозначают действие, которое следует выполнить над данными, например, оператор сложения чисел (+), оператор логического отрицания (!).

7) К **знакам препинания** в C++ относятся два символа: (;) и (,) , служащие для разделения лексем. Символом (;) оканчивается любое предложение (инструкция) программы. Запятая

Элементы языка программирования

3. Выражения

В ходе работы программы могут вычисляться значения **выражений**. **Выражения** образуются по правилам языка программирования. В состав выражений могут входить переменные, соединенные знаками операторов и другие выражения.

Переменные хранят данные, которые обрабатываются в процессе работы программы. Для обозначения переменных используются **идентификаторы**. Каждая переменная имеет определенный **тип**, например, целый, символьный и т.д.

Т.е. **переменная** — именованная область оперативной памяти, предназначенная для хранения значений (данных) определенного типа. Тип влияет на размер памяти и способ внутреннего представления (кодирования) хранимых значений.

Значение выражения можно присвоить какой-либо переменной или непосредственно использовать в дальнейших вычислениях. С помощью круглых скобок можно определять порядок проведения вычислений.

Пример: d= (a+ b) * c;

Здесь можно выделить три выражения: сумма a и b, произведение этой суммы на c и выражение присваивания, создаваемое оператором (=), в результате которого d получает новое значение.

Замечание: выражения присваивания можно использовать как часть других выражений, например,

```
x = (d = (a + b) * c);
```

или

```
x = d = (a + b) * c;
```

Элементы языка программирования

4. Функции

Функция – это подпрограмма, имеющая собственное имя и вызываемая для своего выполнения по этому имени.

Обычно программа на языке C++ состоит из нескольких функций, которые решают отдельные части общей задачи.

Каждая программа должна иметь в своем составе функцию с именем **main**, которая, впрочем, может быть единственной функцией программы.

Выполнение программы начинается именно с вызова функции main (точка входа в прогр.).

5. Комментарии

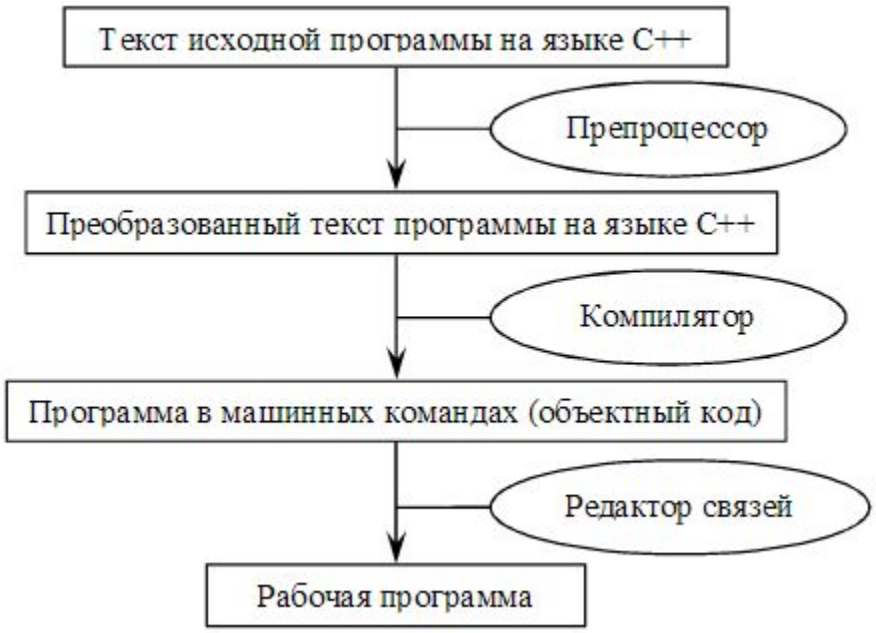
Программы на языке C++ могут содержать **комментарии**, которые никак не влияют на вычисления и служат для пояснения хода работы программы. Комментарии предназначаются не компьютеру, а человеку. В комментариях допустимо использование любых символов, в том числе и русских букв.

Возможны два вида комментариев.

1) Комментарии, занимающие одну строку, начинаются двумя символами наклонной черты // и оканчиваются концом строки.

2) Текст, окаймленный знаками /* и */,также является комментарием. Такой комментарий может быть многострочным.

Процесс создания программы



Программа, написанная на языке программирования, называется исходной. В языке С++ исходная программа может состоять из нескольких файлов. Чтобы исходную программу можно было исполнить, ее нужно перевести на язык машинных команд. Готовая к выполнению программа на языке машинных команд называется рабочей (исполняемой).

Препроцессор осуществляет предварительную обработку исходного текста в соответствии с включенными в него директивами, например, вставляет в текст обрабатываемого файла содержимое других файлов, убирает комментарии, заменяет некоторые слова (называемые макросами) на заранее определенные фрагменты программного кода.

Компилятор проверяет правильность исходной программы с точки зрения правил языка и переводит ее на язык машинных команд, в результате чего создается *объектный код*. Объектный код, полученный после компиляции какого-либо файла, размещается в файлах с расширением .obj.

Редактор связей собирает в один исполняемый файл объектные модули программы, добавляет код стандартных функций из библиотек. В результате получается готовый к выполнению исполняемый код, размещаемый в файле с

Инструменты программирования

I. В настоящее время для разработки программ используют **интегрированные среды разработки** (*Integrated Development Environment* – IDE), включающие в свой состав редактор для написания текста программы, препроцессор, компилятор, редактор связей, а также отладчик, облегчающий поиск ошибок. Примеры IDE для C++:

1) Среда **Eclipse** является бесплатной, свободно распространяемой. По умолчанию эта среда использует язык Java, но есть варианты ее настройки для других языков, в том числе для языков C и C++ . Eclipse не включает компилятор с языка C++, его нужно установить отдельно и указать в настройках среды. Eclipse не удобен для работы на C++.

2) Бесплатная интегрированная среда **Qt Creator**, ориентированная на использование кроссплатформенной библиотеки Qt на C++, предназначенной для разработки приложений для различных операционных систем.

Для разработки на C++ в среде Qt Creator можно использовать свободный компилятор MinGW или компилятор MS к Visual Studio.

3) Компания Embarcadero Technologies выпускает несколько форм среды **Red Studio**, включающей систему **C++ Builder** со средой разработки на C++ и компилятор.

4) Microsoft выпускает продукт **Visual Studio** для Windows, macOC позволяющий разрабатывать приложения на языках C++, C#, Visual Basic и др. для ряда платформ

Существует бесплатная версия Visual Studio Community 2019 для обучения, разработок с открытым кодом и индивидуального использования (https://visualstudio.microsoft.com/ru/)

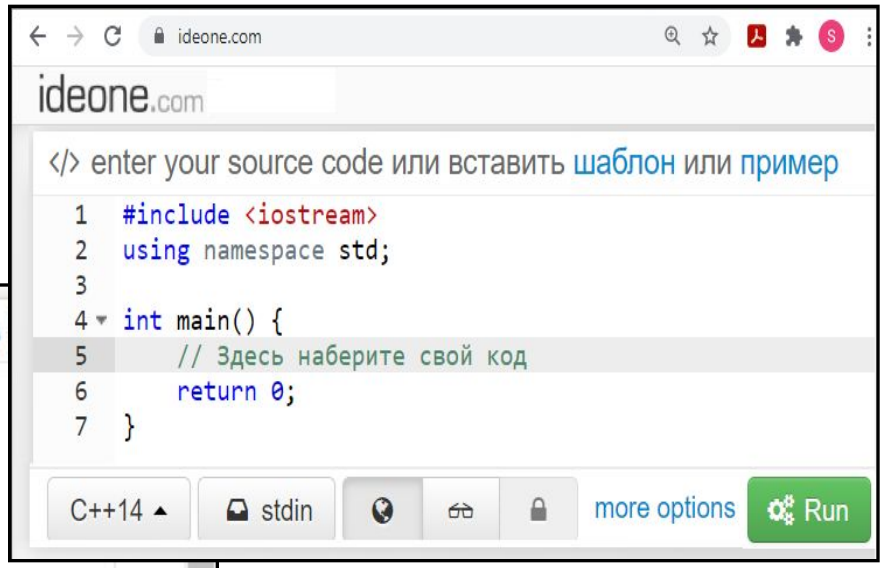
5) Бесплатная среда разработки **Code::Blocks** на C/C++ и Fortran (https://www.codeblocks.org)

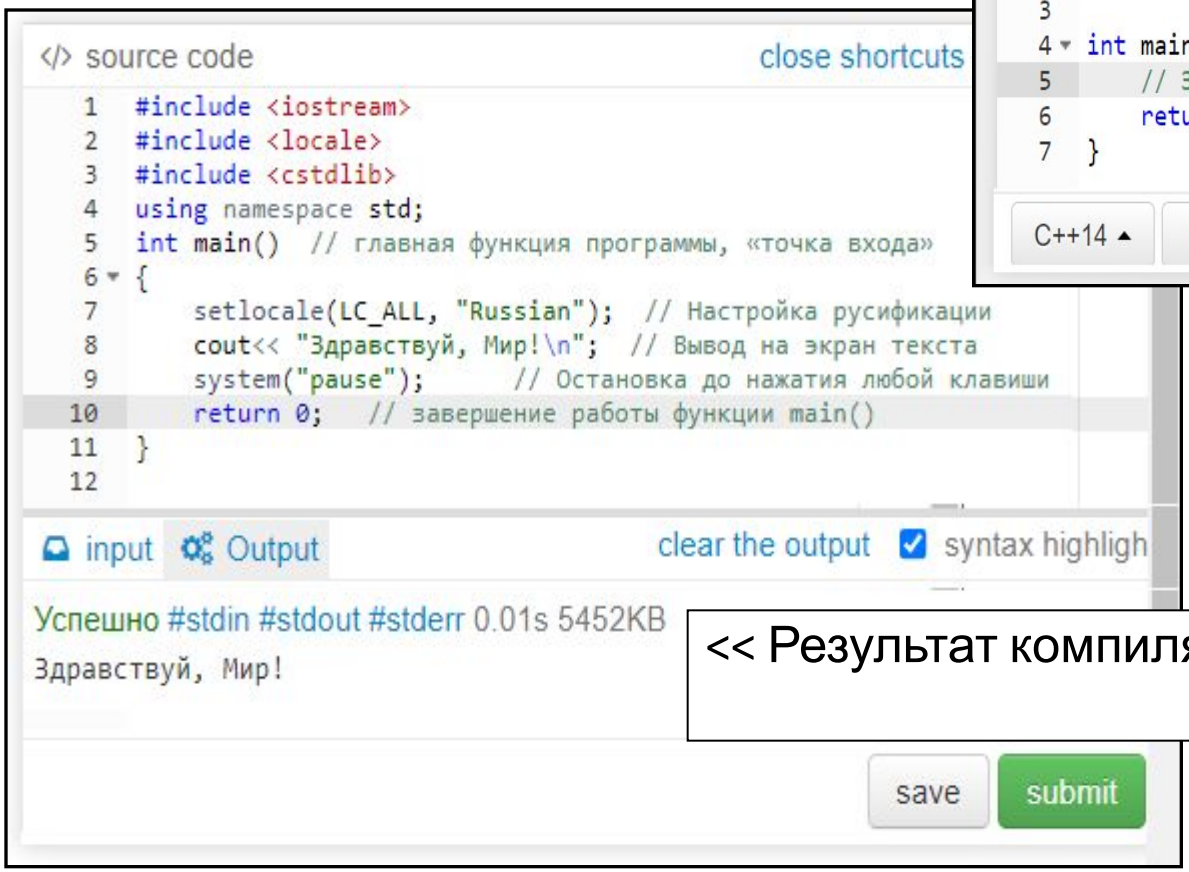
II. Online компиляторы с C++

(см., например, краткий обзор на https://purecodecpp.com/archives/3902)

Инструменты программирования

Вид страницы **ideone.com** с настройкой на ввод, компиляцию и запуск программы на C++ >>>>





<< Введенная на **ideone.com** программа C++

<< Результат компиляции, сборки и выполнения программы

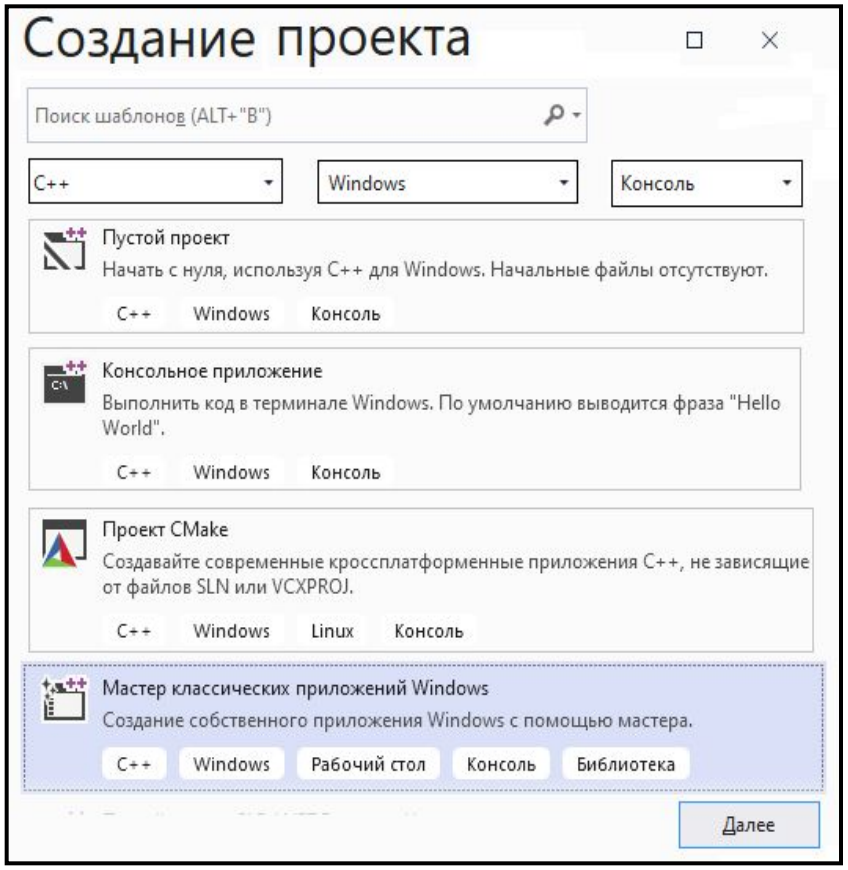
Замечание. *Приводимые далее примеры программ проверены в среде Visual Studio.*

Первая программа

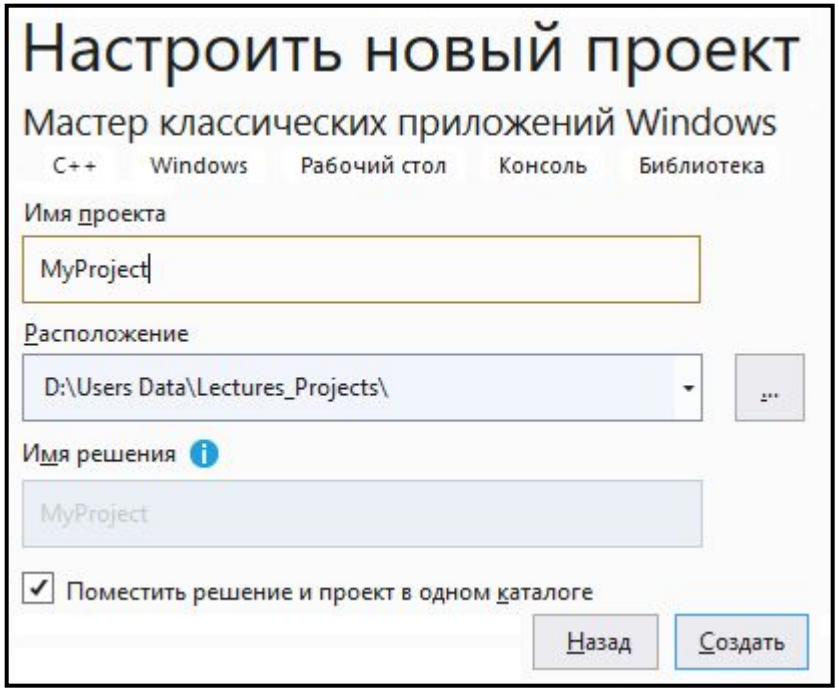
```
// Файл Hello.cpp
#include <iostream>          // Подключение заголовочного файла  библиотеки стандартных
                             // потоковых средств ввода и вывода  языка C++
#include <locale>            // Подключение заголовочного файла  библиотеки, содержащей
                             // средства для настройки программы на использование
                             //особенностей, характерных  для разных стран и языков, в
                             //частности, русского языка
#include <cstdlib>           // Подключение заголовочного файла  библиотеки, содержащей
                             //функцию system()
using namespace std;         //делает доступными имена из пространства имен std
int main()                   // главная функция программы, «точка входа»
{
    setlocale(LC_ALL, "Russian");        // Настройка на использование национальных
                                         //особенностей
    cout << "Здравствуй, Мир!\n";       // Вывод на экран текста Здравствуй, Мир!
    system("pause");                     // Остановка программы до нажатия
                                         // любой клавиши
```

Создание пустого проекта

Загружаем среду и выполняем команду меню **Файл=>Создать=>Проект**... Далее в окне **Создание проекта** выбираем язык программирования (**C++**), ОС (**Windows**), тип проекта (**Консоль**) и шаблон проекта (**Пустой проект** или **Мастер классических приложений Windows**).



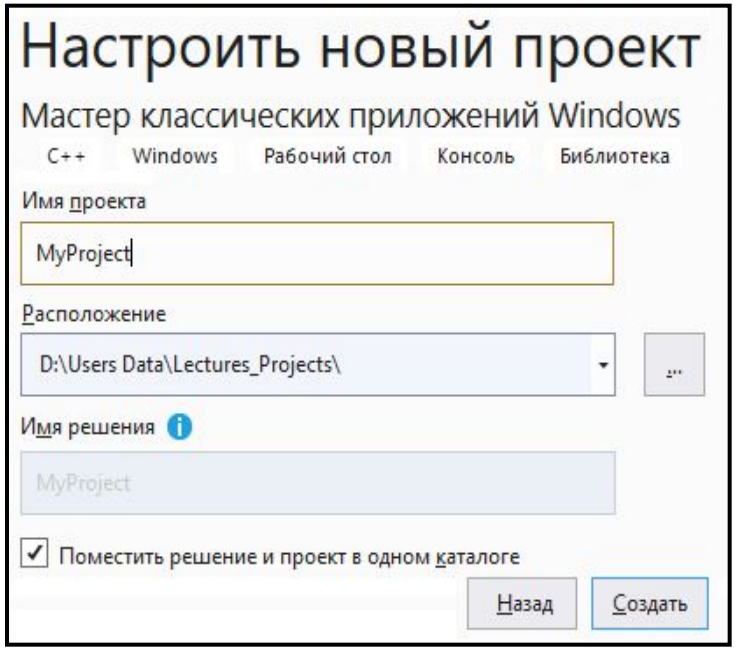
После нажатия кнопки **Далее** откроется окно мастера создания приложения.



В поле **Имя** вводим имя проекта, в поле **Расположение** выбираем папку для проекта. При создании проекта автоматически создается папка с именем, совпадающим с именем проекта. Если включить флаг «*Поместить решение и проект в одном каталоге*» (удобно для небольших программ), дополнительных

Создание пустого проекта

Когда в Visual Studio ведется работа с несколькими взаимосвязанными проектами, то их удобно объединять в так называемые **решения** (**Solution**). В этом случае флаг «*Поместить решение и проект в одном каталоге*» включать не нужно. Тогда поле **Имя решения** будет доступно для редактирования. В нем можно будет задать имя решения, отличающееся от имени проекта, а для решения будет создана отдельная папка с этим именем, в нее будут вложены папки всех проектов, входящих в состав **Solution**.



При создании простых учебных программ, включающих один проект, удобнее использовать режим, когда отдельной папки для решения не создается. В этом случае его папка совпадает с папкой проекта.

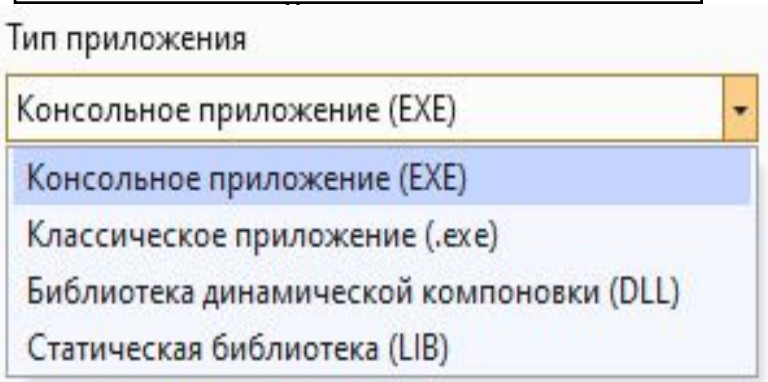
Этот режим как раз и включается установкой флага «*Поместить решение и проект в одном каталоге*».

В любом случае в папке решения создается **файл решения** с расширением **.sln**. В папке каждого проекта создаются **файлы проектов** с расширением **.vcxproj**

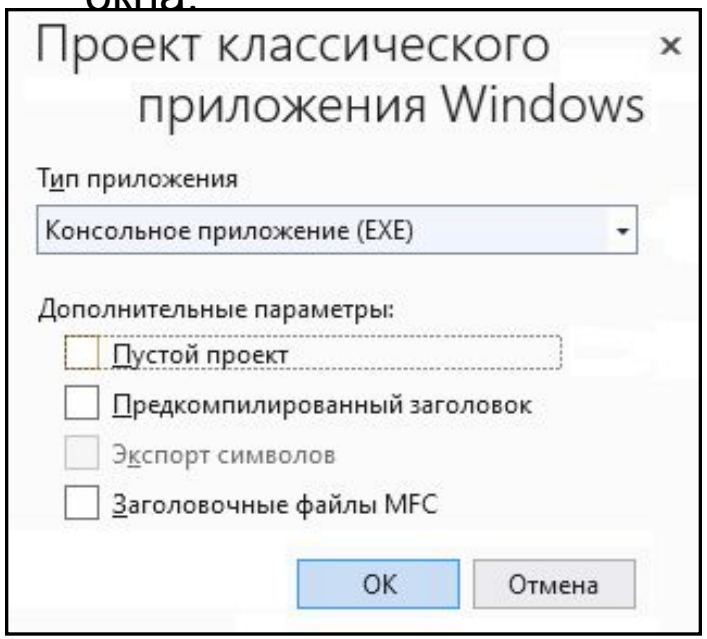
Создание пустого проекта

При нажатии кнопки **Создать** в окне «Настроить новый проект» попадаем в диалог «Проект классического приложения Windows», где нужно выбрать тип приложения. ====>>>>

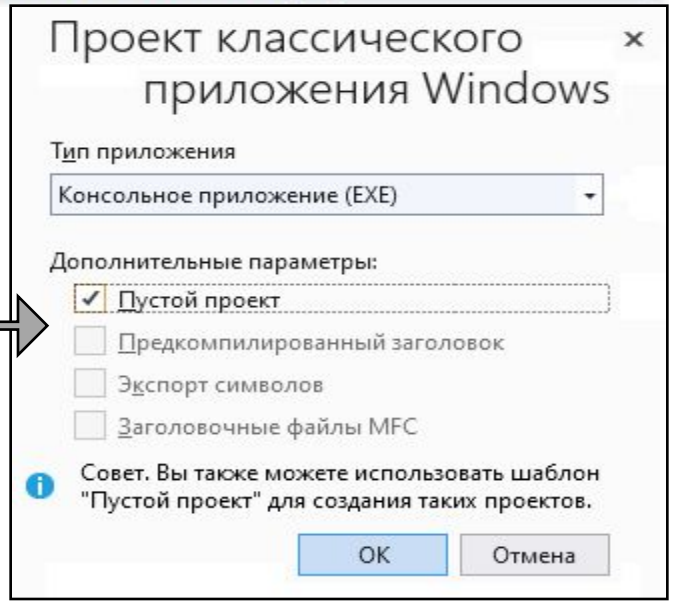
Возможные типы



Вид диалогового окна:



Для создания **пустого проекта** выбираем тип «Консольное приложение (EXE)», помечаем «Пустой проект», подтверждаем по «ОК».



Для непустых проектов доступны к выбору два флага: «*Предкомпилированные заголовки*» и «*Заголовочные файлыMFC*».

Выбор **MFS** – встраивает в проект поддержку библиотеки MFC.

Консольные приложения выполняются в отдельном окне, эмулирующем работу операционной системы

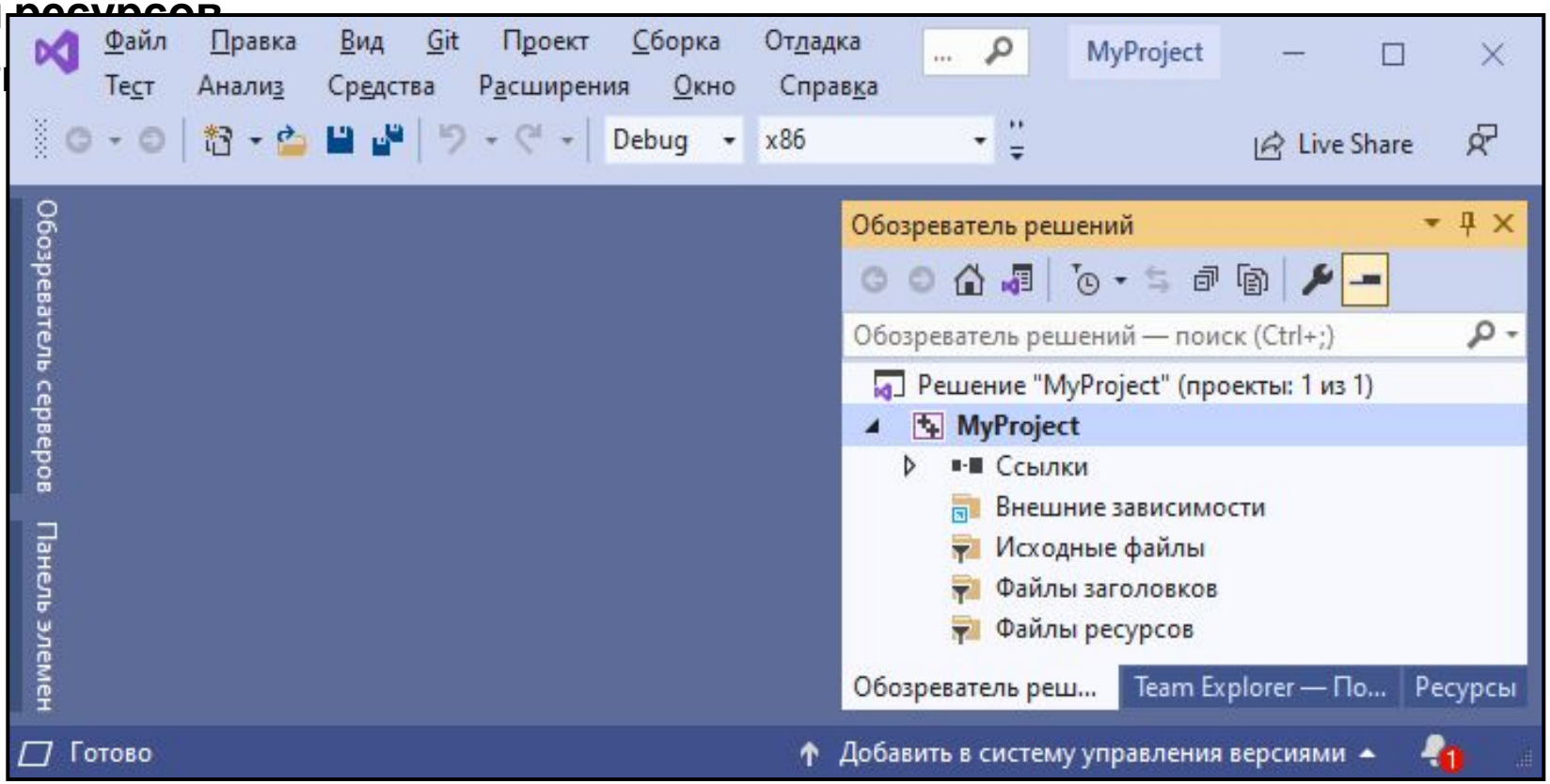
Создание пустого проекта

Нажатием кнопки **ОК** завершается создание проекта и открывается среда разработки.

В окне **Обозреватель решений** показывается состав решения. Видно, что решение MyProject содержит один проект с тем же названием, включающий три основных раздела:

- **Исходные файлы**;
- **Файлы заголовков**;
- **Файлы ресурсов**.

Пока эт



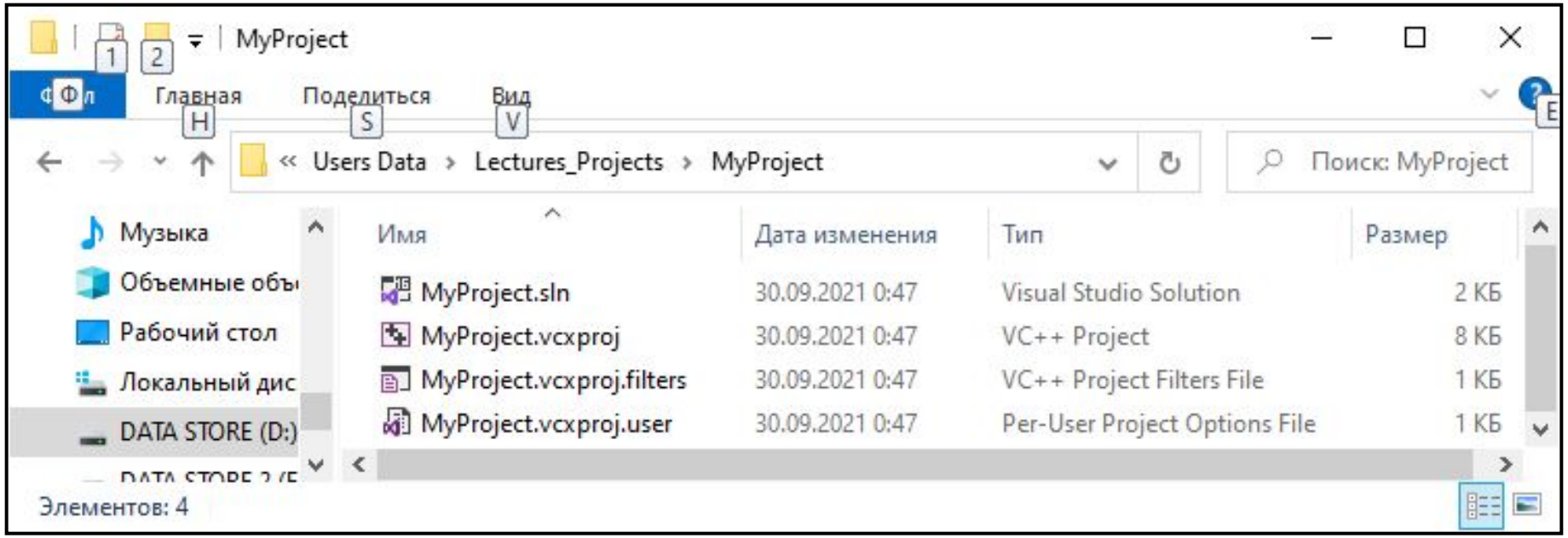
Создание пустого проекта

Содержимое папки созданного пустого проекта в Проводнике Windows.

Файл проекта MyProject**.vcproj** содержит информацию о версии Visual C++, которая сгенерировала этот файл, информацию о платформе, конфигурациях и свойствах проекта, включая параметры компиляции, сборки и пр.

Файл решения MyProject.**sln** содержит сведения о проектах, входящих в данное решение.

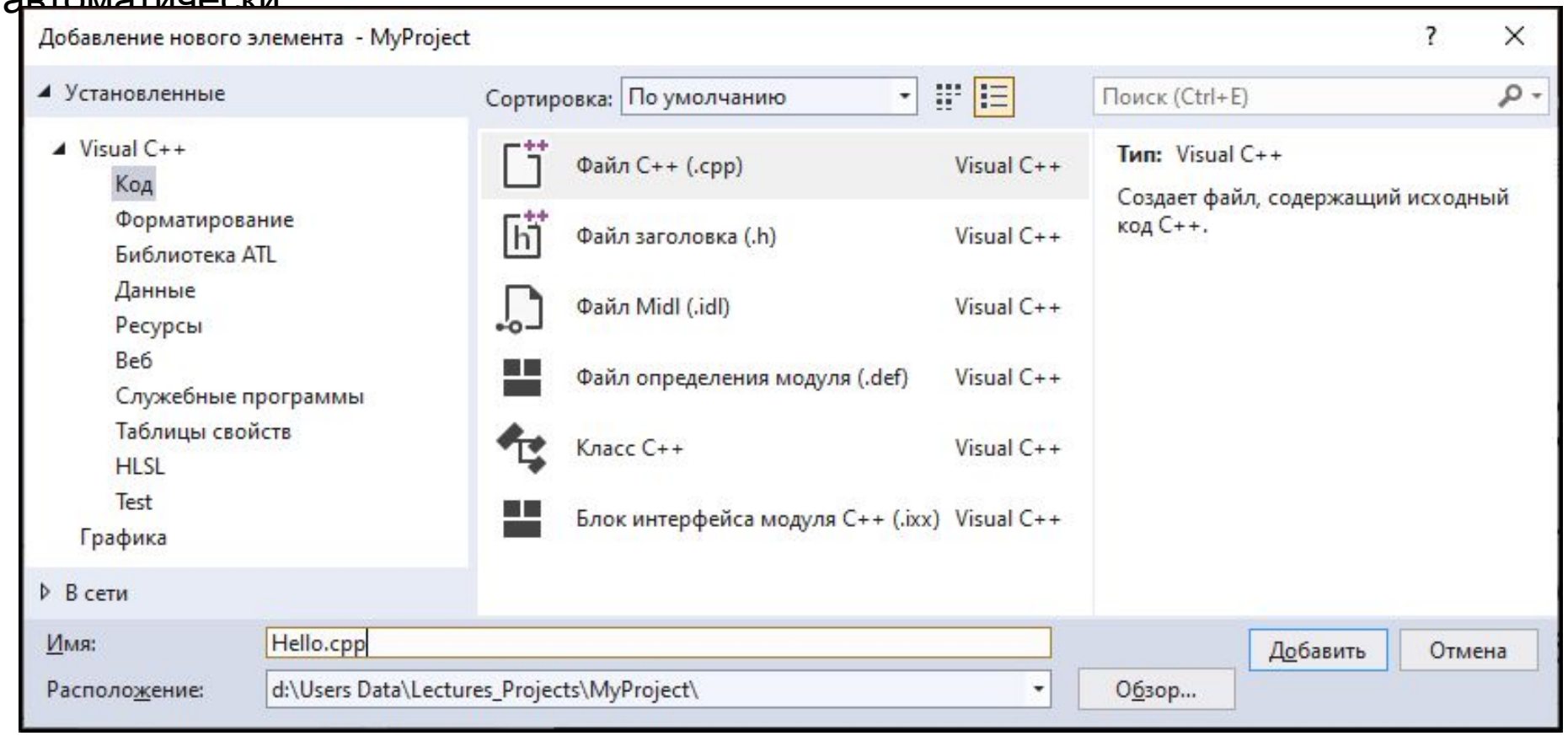
Остальные файлы имеют служебный характер для среды разработки.



Использование пустого проекта

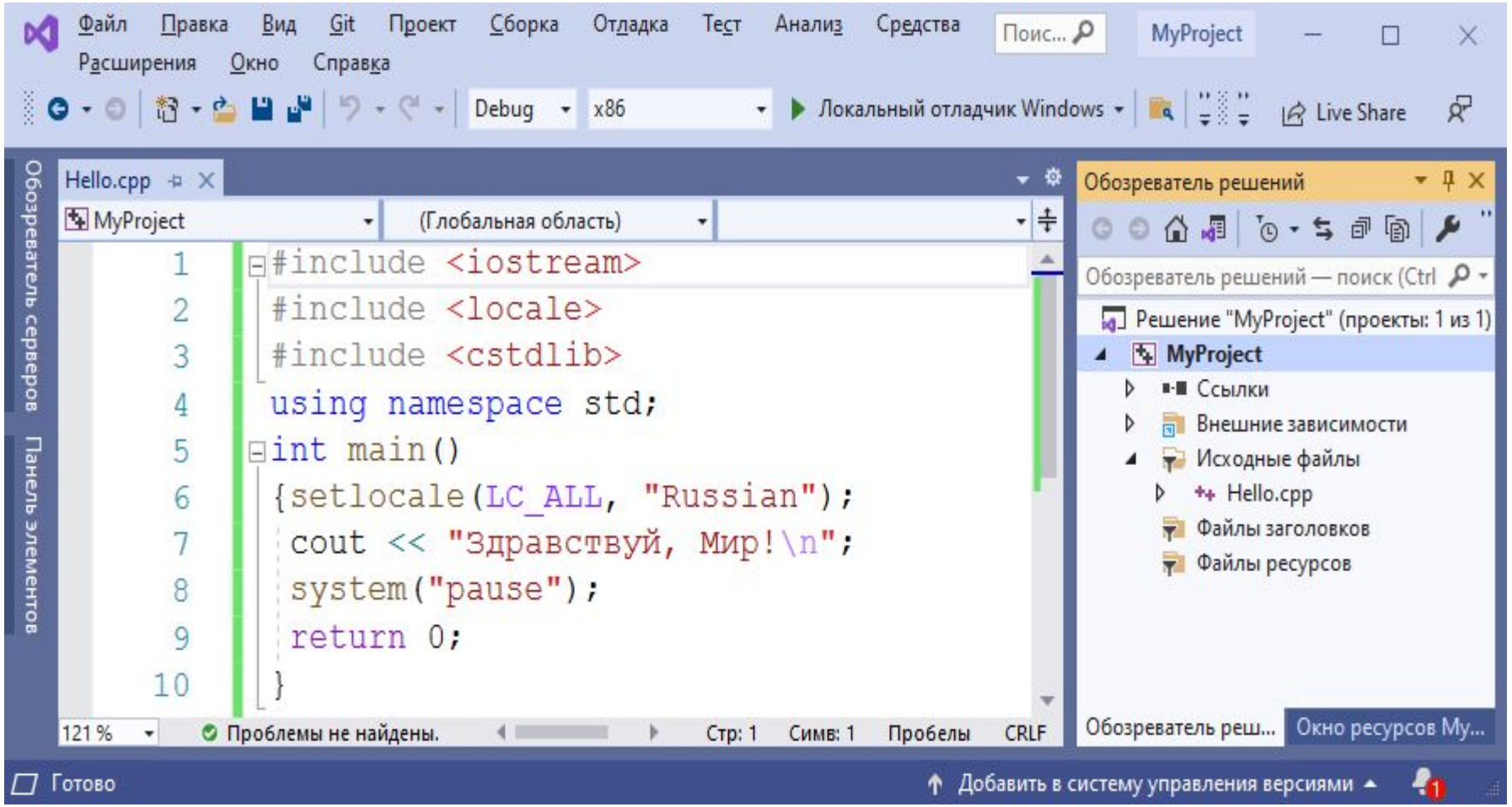
Добавление в проект файла исходного кода.

Выполним команду **Проект => Добавить новый элемент…**. В появившемся окне **Добавление нового элемента – MyProject** выбираем категорию – **Код**, шаблон – **Файл C++ (.cpp)**. В поле **Имя** вводим имя добавляемого файла кода – Hello.cpp Расширение .cpp можно не писать – будет добавлено к имени файла автоматически



Использование пустого проекта

В появившемся пустом окне текстового редактора наберем или вставим код Программы 1 со слайда 9.

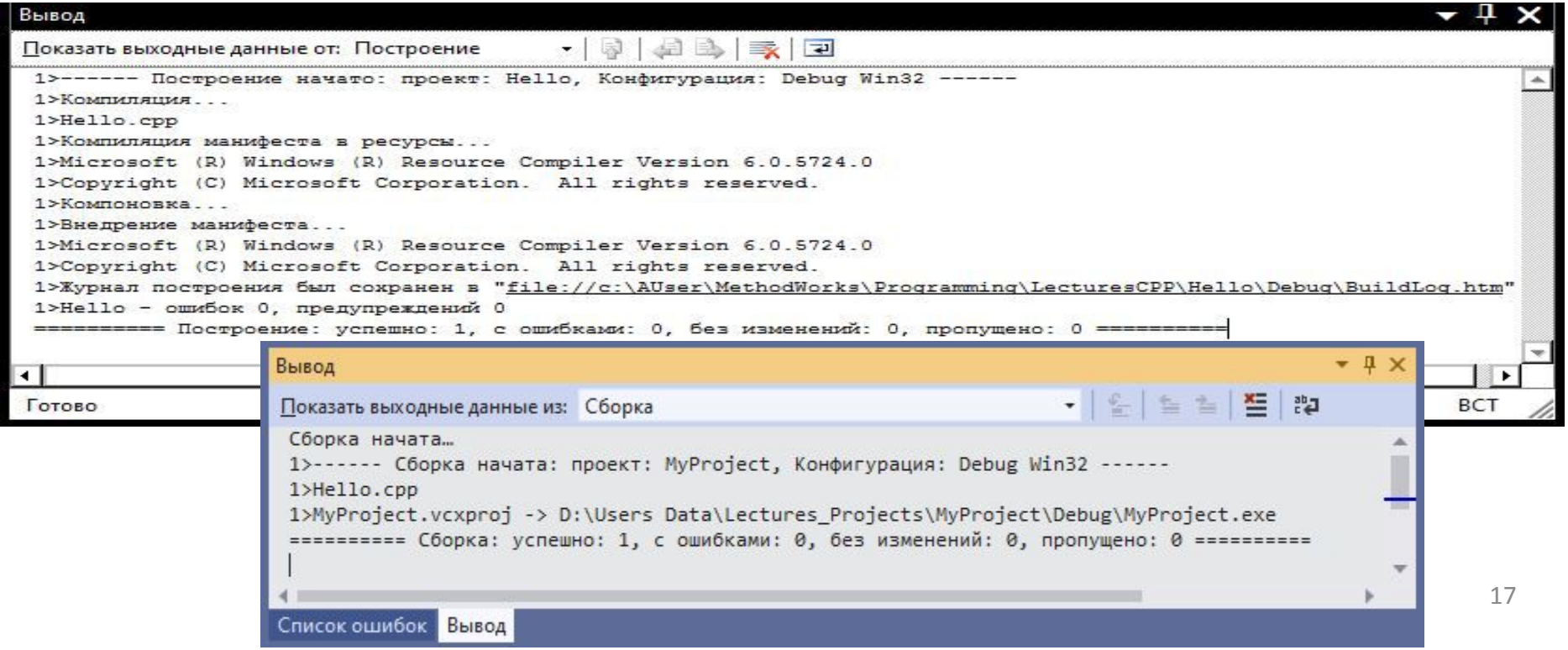


Построение проекта

Построение рабочей программы выполняется командой **Сборка => Построить MyProject**. При первом выполнении этой команды сначала запускается компилятор. Если при компиляции не обнаружено ошибок, запускается компоновщик.

Сообщения в окне **Вывод** зависят от версии Visual Studio. Например, в окне **Вывод** могут быть показаны сообщения: *1>Компиляция... з*атем – *1>Компоновка...* . Или же: *Сборка начата…* Если компилятор или компоновщик находят ошибки, сообщения о них также выводятся в окно **Вывод**.

Для более удобной работы с ошибками существует окно **Список ошибок**, выводимое командой **Вид => Список ошибок**.



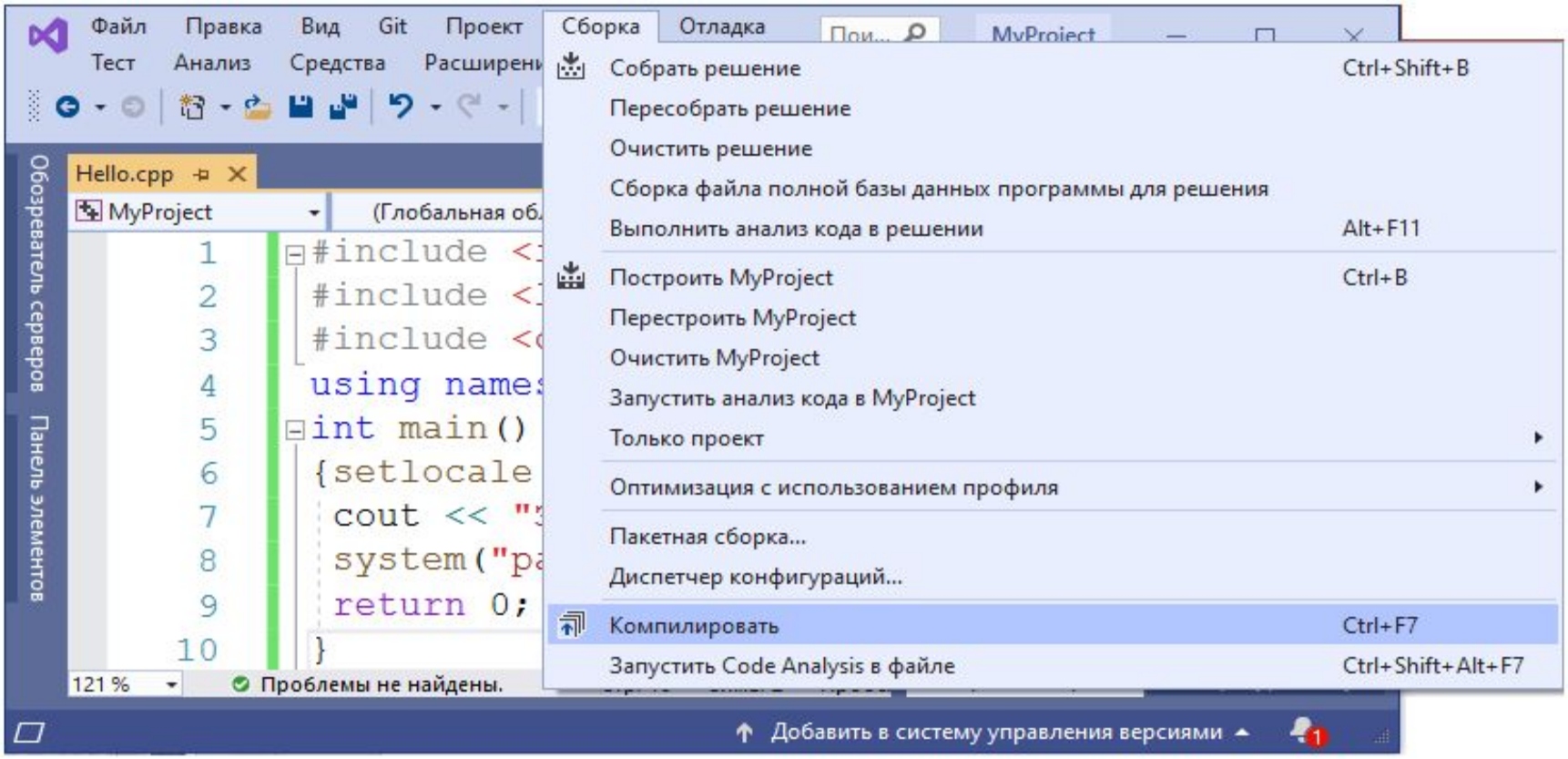
Построение проекта

Если ошибок не было, в окне **Вывод** будет напечатано: **Построение: успешно: 1**.

При повторном выполнении команды **Построить <Имя_Проекта>** компилируются только файлы, в которые внесены изменения, и, если в исходных файлах были изменения, выполняется компоновка.

Команда **Сборка => Перестроить <Имя_Проекта>** выполняет перекомпиляцию всех файлов проекта и компоновку.

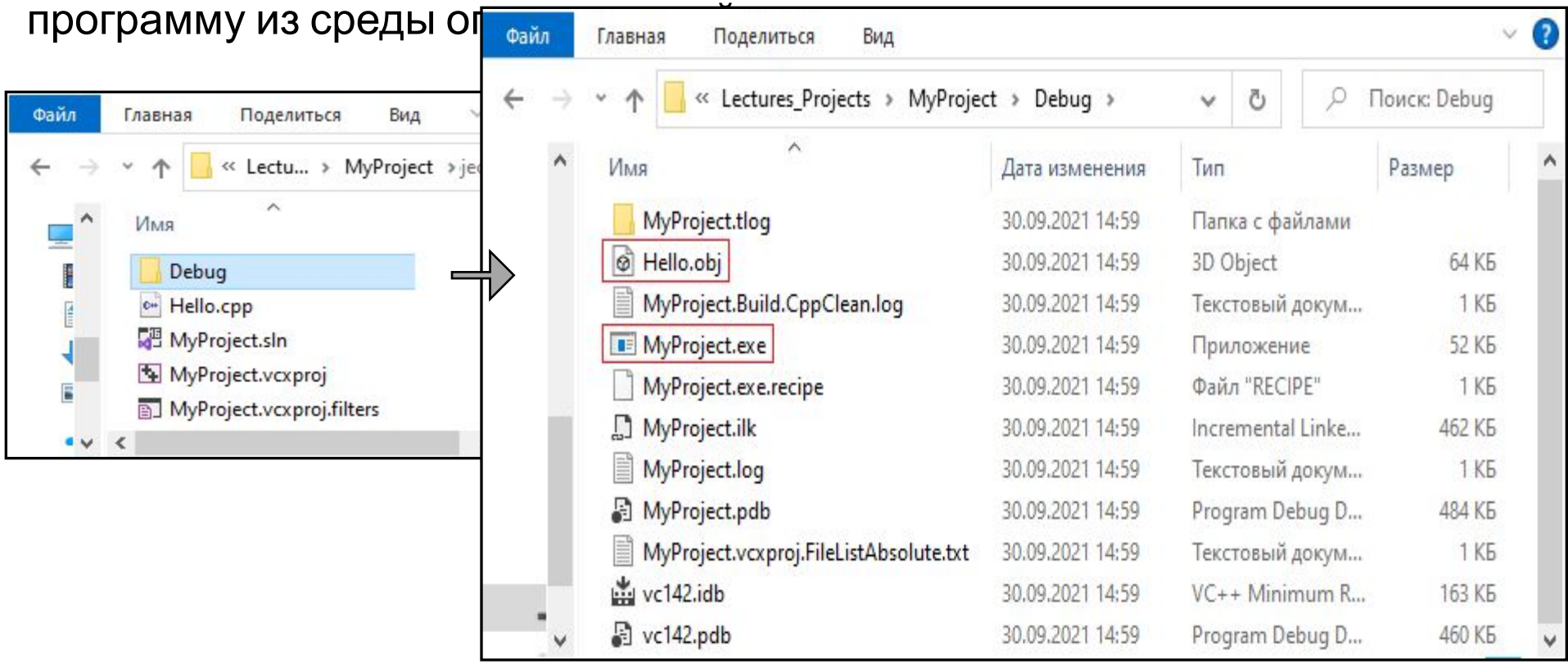
Отдельная компиляция выделенного файла выполняется командой **Сборка => Компилировать** или комбинацией **Ctrl+F7**.



Построение проекта

При построении приложения в режиме отладки в папке проекта создается папка **Debug**, содержащая результаты работы компилятора и компоновщика.

В этой папке мы видим объектный файл программы *Hello.obj* и рабочую программу *MyProject.exe*. Используя созданный exe-файл, можно запускать программу из среды от



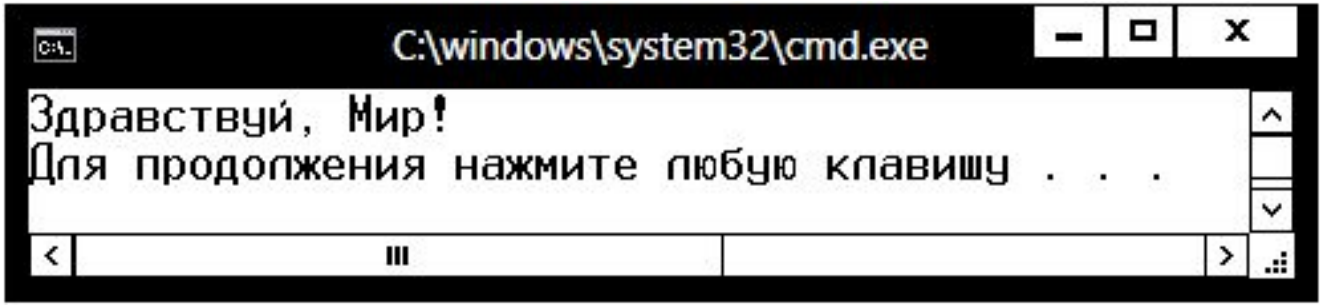
Выполнение команд **Сборка => Очистить** <*Имя_проекта*> приведет к удалению файлов, возникших в процессе компиляции и сборки.

Запуск программы

Из среды разработки программу можно запустить в режиме отладки командой **Отладка => Начать отладку**, или клавишей **F5**. При этом окно выполнения программы сразу закрывается, после окончания работы программы.

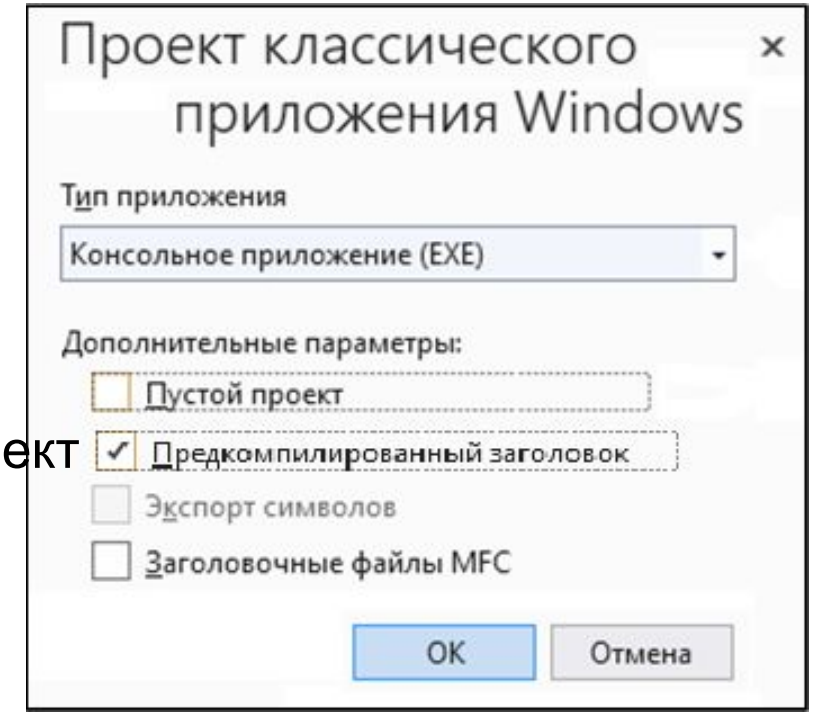
После команды **Отладка => Запуск без отладки** (комбинация клавиш **Ctrl+F5**) окно выполнения программы остается открытым до нажатия любой клавиши.

Примечание. Включение в программу перед завершающей инструкцией **return 0**; инструкции `system("pause");` приведет к дополнительной приостановке выполнения программы до нажатия любой клавиши. Вот как будет выглядеть консольное окно приложения при запуске в режиме **Отладка => Начать отладку** при наличии перед завершением инструкции `system("pause")`.



Создание непустого проекта

Создадим проект NotEmptyPr. При назначении параметров приложению *НЕ уста-навливаем* флажок **Пустой проект.** Полезно установить флажок **Предкомпилированный заголовок.**



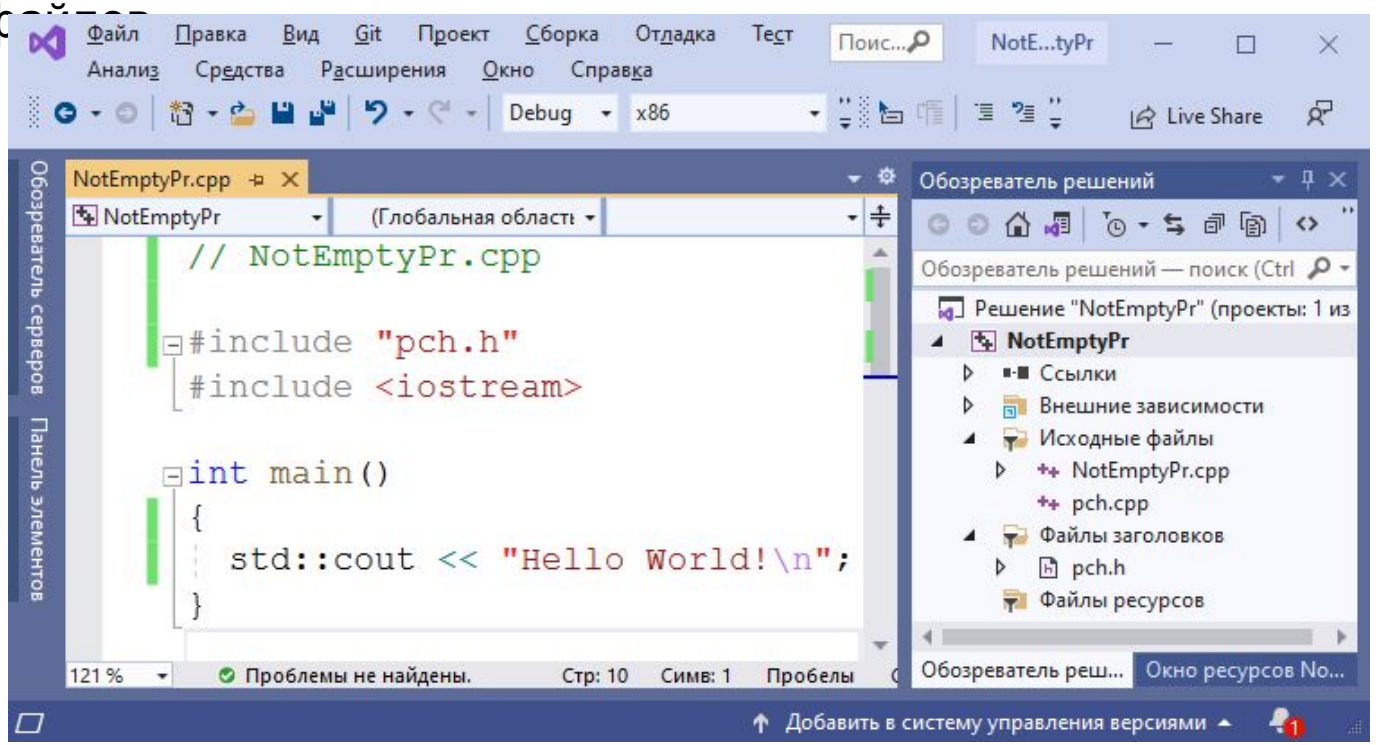
После нажатия кнопки **ОК** создается проект указанного типа и открывается в среде разработки.

При *НЕ установленном* флажке **Пустой проект** в проект добавляется файл исходного кода с именем, совпадающим с именем проекта, который будет содержать заготовку функции main().

Создание непустого проекта

В окне **Обозреватель решений** видно, что в состав проекта автоматически включено несколько файлов.

Файл NotEmptyPr.cpp, содержит заготовку главной функции программы main()



Файлы **pch.h**, **pch.cpp** определяют состав предварительно компилируемых заголовочных файлов (*precompiled header*, *PCH*). Программист может включить в **pch.h** директивы #include <имя_заголовка> для подключения набора редко изменяемых больших заголовочных файлов.

При первом построении проекта, выполняемого командой **Построение=>Построить <ИмяПроекта>**, создается файл предварительно откомпилированных заголовков NotEmptyPr.pch и файл pch.obj – с результатами компиляции заголовочных файлов, включенных программистом в pch.h

Создание непустого проекта

Файл pch.h

```
// pch.h: это файл предварительно компилируемых заголовков.
// Подключенные в области с пометкой @@@@@@ файлы заголовков
//  компилируются только один раз, что ускоряет последующие сборки.
// Результаты предкомпиляции запоминаются в файле ИМЯ_ПРОЕКТА.pch
// Однако изменение любого из приведенных здесь файлов
//        между операциями сборки приведет к повторной
//             компиляции всех(!) этих файлов.
// Не добавляйте сюда файлы, которые планируете часто изменять,
//  так как в этом случае выигрыша в производительности не будет.
#ifndef PCH_H
#define PCH_H
// @@@@@@ Добавьте сюда заголовочные файлы
//                для их предварительной компиляции
#endif //PCH_H
```

pch.h – файл для подключения стандартных системных заголовочных файлов или заголовочных файлов конкретного проекта, которые часто используются, но не часто изменяются

Создание непустого проекта

Файл pch.cpp

```
// pch.cpp: файл исходного кода,
//  парный к файлу предварительно компилируемых заголовков
#include "pch.h"
// Для использования механизма предварительно скомпилированных
//      заголовочных файлов файл pch.cpp необходим
//       среди исходных файлов проекта
// Подключайте ссылки на требующиеся дополнительные заголовки
//           в файле pch.h, а не в файле pch.cpp
```

Замечание: в файл pch.cpp не следует вносить каких-либо дополнений.

Программа «Непустой проект»

Файл **NotEmptyPr.cpp**

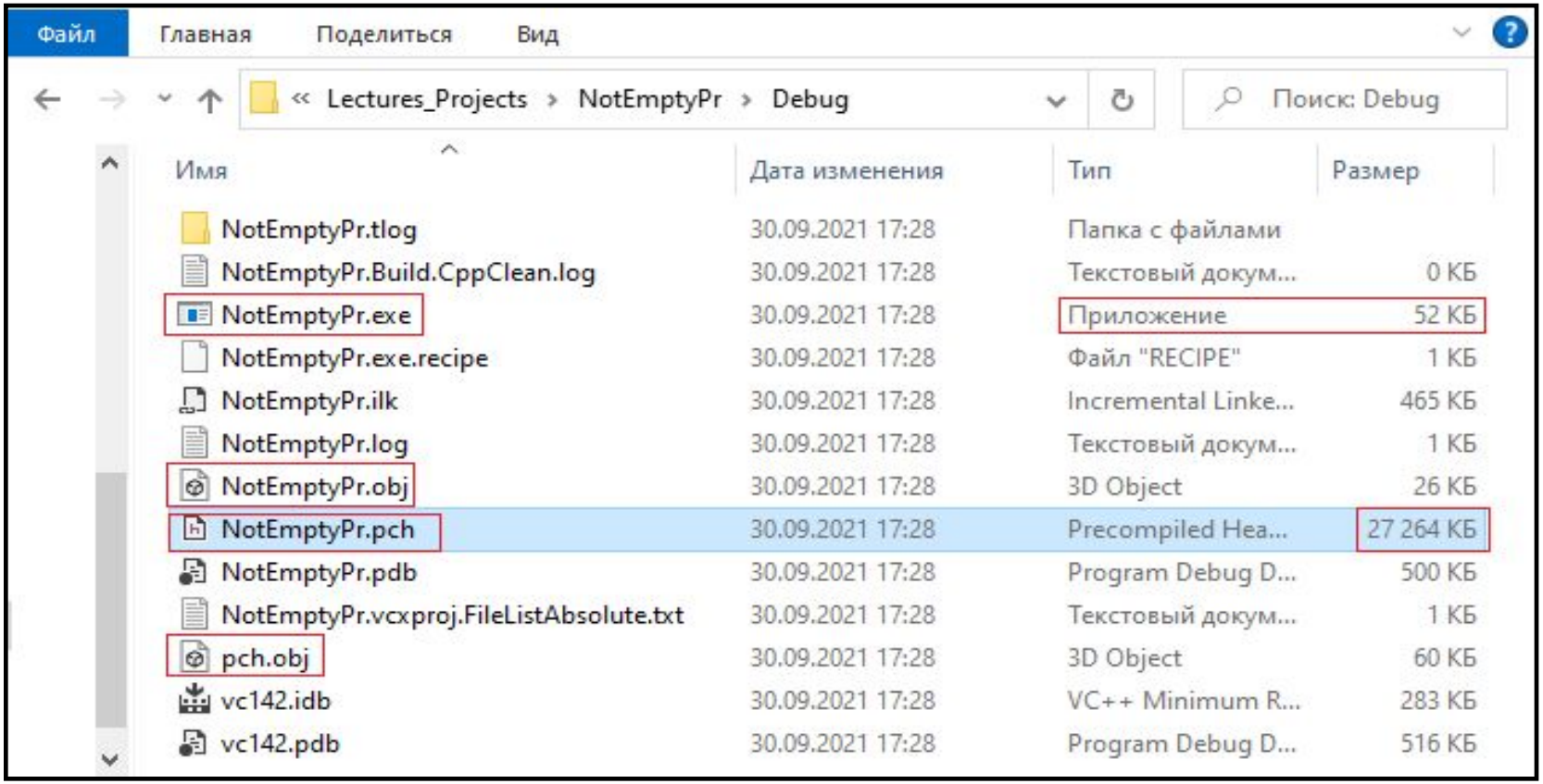
```
#include "pch.h"
// Три следующих ниже заголовочных файла правильнее
//        перенести в файл pch.h:
#include <iostream>
#include <locale>
#include <cstdlib>
// using namespace std; //— отказались от подключения пространства имен std
// Изменим шаблонную функцию main() в файле NotEmptyPr.cpp:
int main()
{  setlocale(LC_ALL, "Russian");
   std::cout << "Здравствуй, Мир!\n"; //Требуется явное указание пространства имен std
   system("pause");
    //   return 0;  // – в функции main инструкция «return 0;» не обязательна
}
```

Перенос в заголовочный файл pch.h стандартных файлов заголовков существенно увеличит объем файла предкомпилированных заголовков **NotEmptyPr.pch**, но ускорит последующие перекомпиляции проекта.

Программа «Непустой проект»

При первом построении непустого проекта в папке проекта создается папка **Debug**, содержащая результаты работы компилятора и компоновщика. В этой папке находится файл предварительно откомпилированных заголовков **NotEmptyPr.pch**, объектный файл **pch.obj**, объектный файл главного модуля **NotEmptyPr.obj** и рабочая программа **NotEmptyPr.exe**.

Для запуска созданной программы из среды операционной системы нужно выбрать созданный exe-файл и нажать клавишу **Enter**.



Отладка программ

Программа «Деление чисел»

Рассмотрим программу, в которой есть синтаксическая ошибка (пропущена тоска с запятой) и семантическая ошибка (деление на нуль).

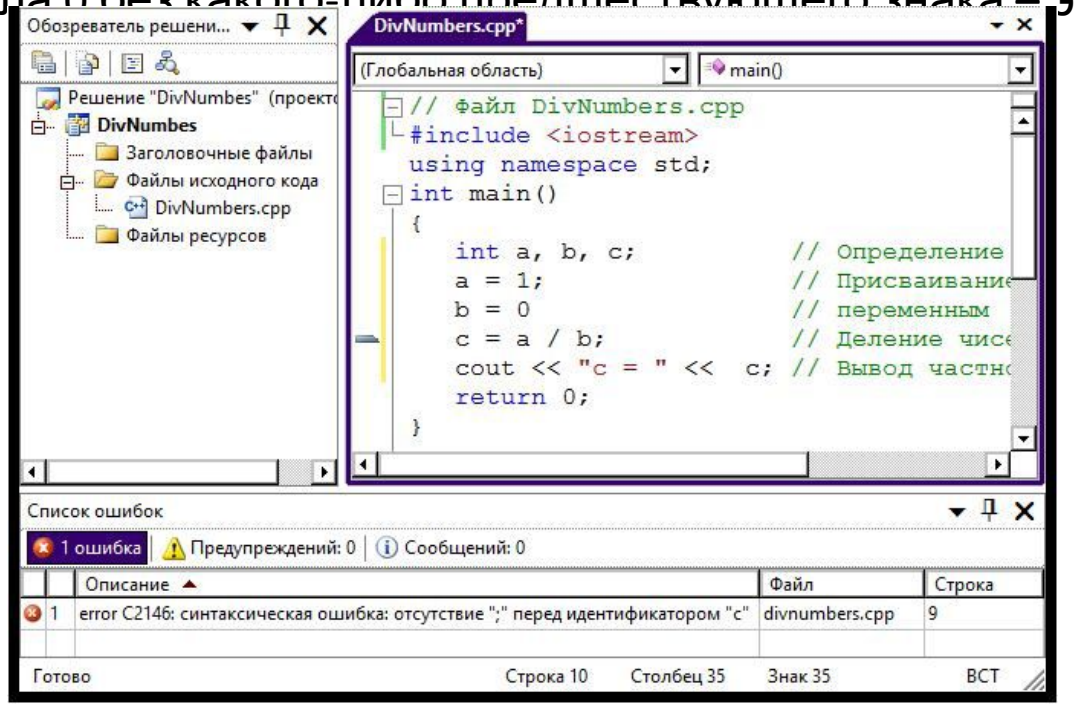
```
// Файл DivNumbers.cpp
#include <iostream>
using namespace std;
int main()
{
    int a, b, c;     // Определение переменных
    a = 1;           // Присваивание значений
    b = 0            // переменным В строчке ошибка, отсутствует  (;)
    c = a / b;       // Деление чисел
    cout << "c = " << c; // Вывод частного
    return 0;
}
```

Отладка программ
Синтаксические ошибки

Выполним компиляцию, нажав **Ctrl+F7**. Компилятор выдаст сообщение об ошибке:

Ошибка 1 error C2146: синтаксическая ошибка: отсутствие ";" перед идентификатором "c".

Компилятор указал на 9-ю строку программы, хотя символа (;) не хватает в конце 8-й строки. Это объясняется тем, что компилятор анализирует текст программы последовательно, символ за символом, выделяя отдельные слова (лексемы) программы. Число 0 в конце 8-й строки ошибкой не является, но имя переменной c, стоящее после числа 0 без какого-либо предшествующего знака – это ошибка.

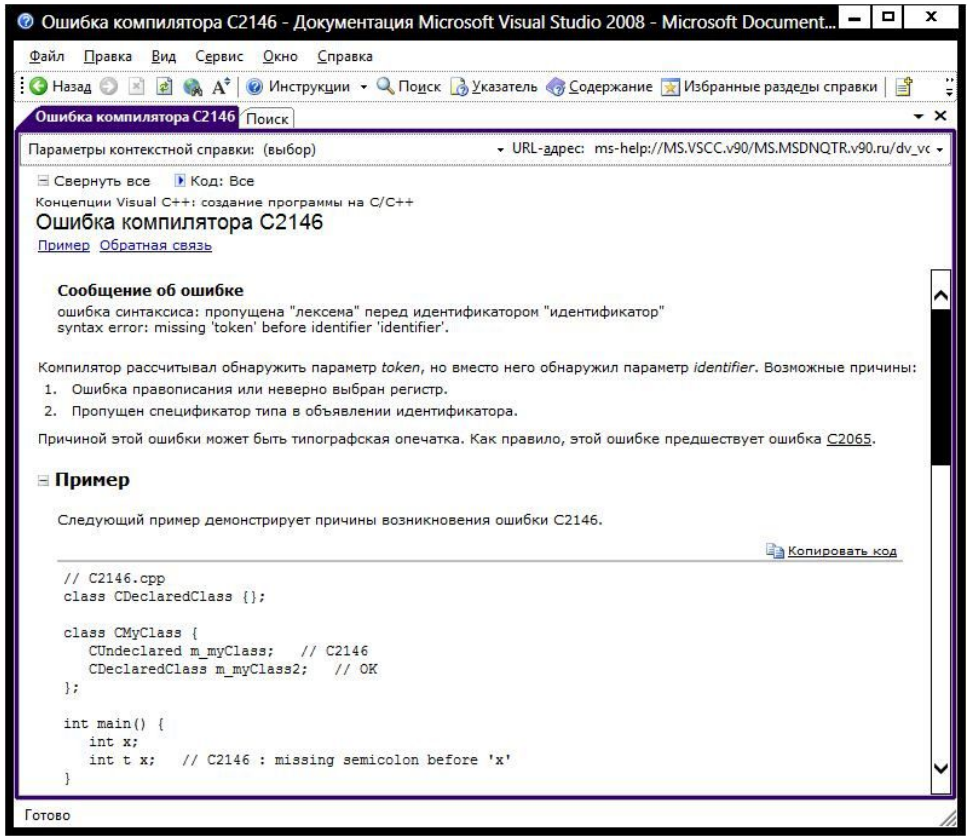


Отладка программ

Синтаксические ошибки

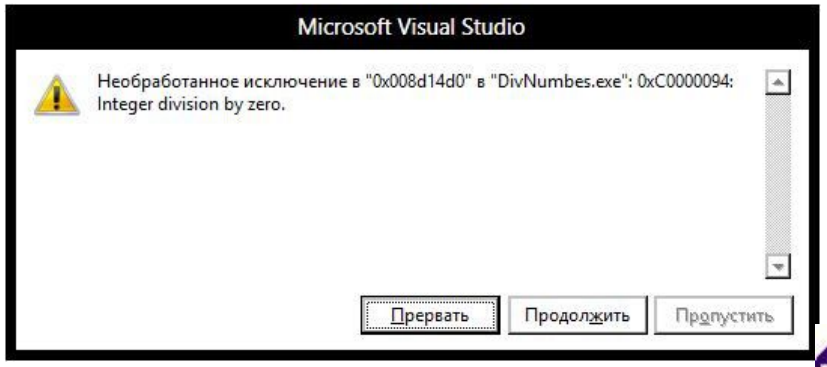
Для получения подробного описания ошибки можно ее выделить и нажать **F1**.

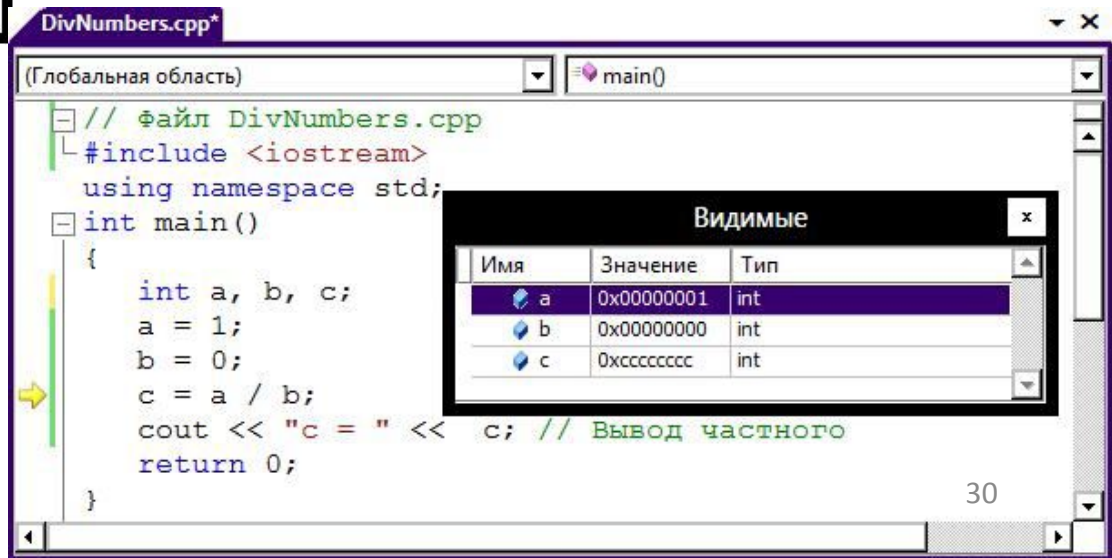
Кроме сообщений об ошибках компилятор может выдавать **предупреждения**. Так выдается предупреждение, если созданная переменная не используется. Наличие предупреждений не препятствует созданию рабочей программы, но лучше их устранять, например, удалить из программы переменную, которая не используется.



Ошибки на стадии выполнения программы

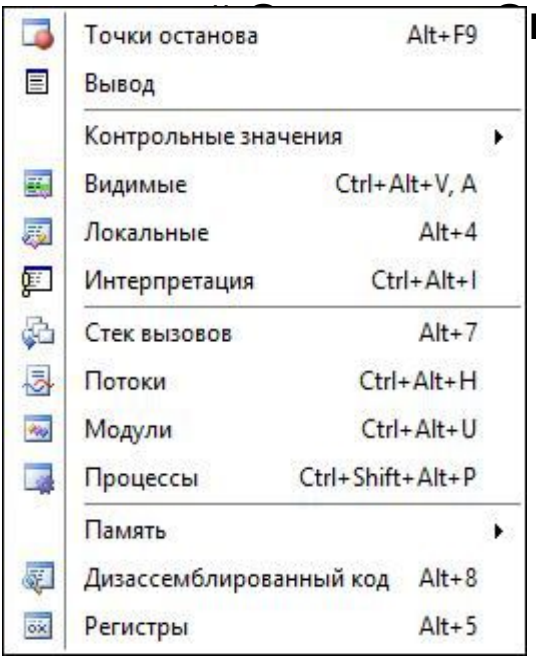
Запустим программу на выполнение командой **Отладка=>Начать отладку**, или нажав **F5**. Появится окно с сообщением Integer division by zero (Целочисленное деление на нуль). Нажмем в этом окне кнопку **Прервать**, перейдем в режим отладки. При этом откроется окно с исходным кодом, в котором желтой стрелкой будет отмечена строка кода, на которой программа остановилась.





Ошибки на стадии выполнения программы

В режиме отладки можно использовать различные окна, помогающие проанализировать работу программы. Список отладочных окон выводится

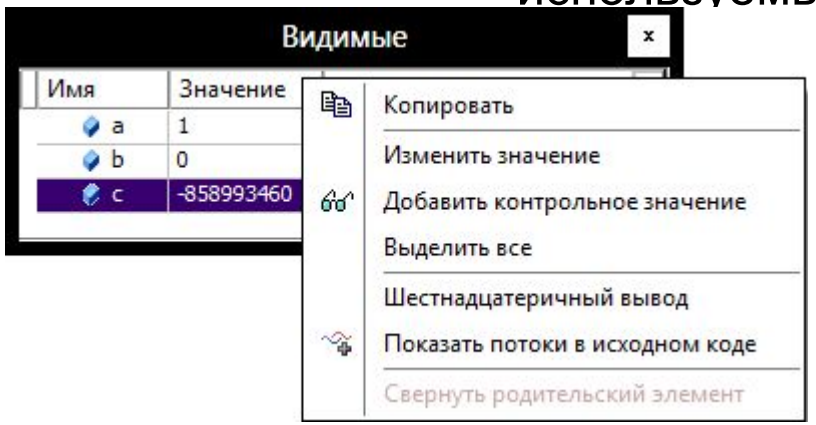


Контрольные значения – окна для наблюдения за текущими
значениями переменных;

Видимые – окно, показывающее переменные, используемые в текущей строке кода и в предыдущей строке кода;

Локальные – окно, в котором автоматически показываются текущие значения локальных переменных;

В окне **Видимые** показаны переменные a, b и c, используемые в строке, на которой остановилась



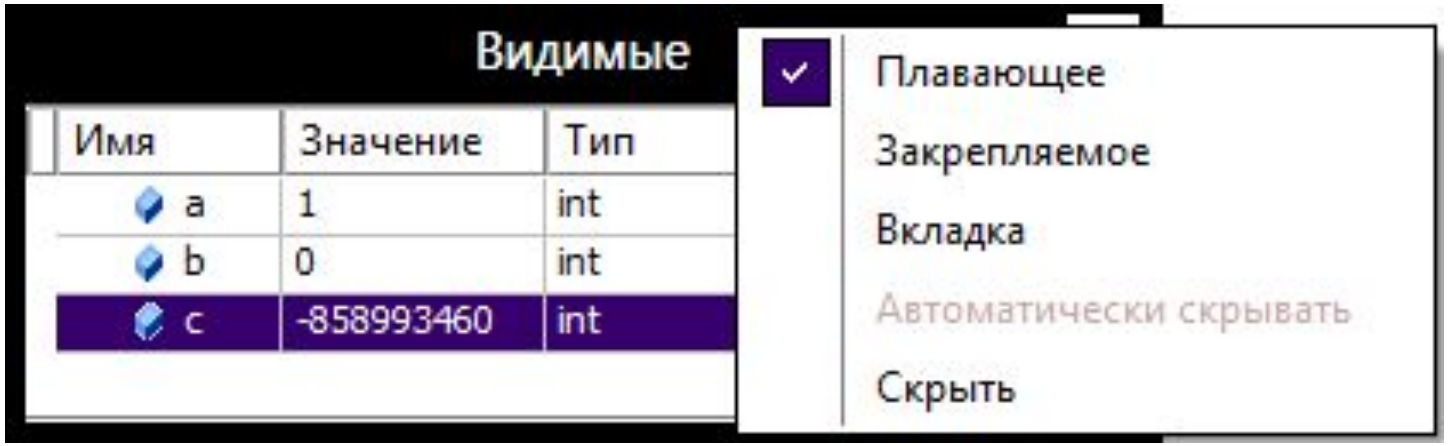
Значения переменных показаны как шестнадцатеричные числа. Для отображения чисел в десятичной системе нужно щелкнуть правой кнопкой мыши по столбцу **Значение** и снять флажок **Шестнадцатеричный вывод**

Ошибки на стадии выполнения программы

В режиме отладки текущие значения переменных отображаются также при наведении на них курсора.

Для управления размещением отладочными и другими окнами среды Visual Studio можно использовать команды контекстного меню, выводимые щелчком правой кнопки мыши по заголовку окна.

В режиме **Плавающее** окно может занимать произвольное положение на экране. В режиме **Закрепляемое** окно можно прикрепить к любой из четырех сторон основного окна. В режиме **Вкладка** окно становится вкладкой основного окна.



Пошаговое выполнение программы

Построчное выполнение программы осуществляется нажатием клавиши **F11** (или командой меню **Отладка=>Шаг с заходом**). Каждое нажатие **F11** приводит к выполнению одной строки кода. Выполняя по шагам программу, обнаруживаем, что ее выполнение прерывается при достижении строки `c = a / b;`

При нажатии **F11** происходит «заход» в функцию, если в выполняемой строке есть обращение к функции.

Команда **Отладка=>Шаг с обходом** или **F10** также осуществляет построчное выполнение программы, но без захода в функции.

Команда **Отладка=>Остановить отладку** (**Shift+ F5**) прерывает отладку. В тексте программы можно поместить точки останова в тех строках, которые вызывают подозрение в правильности работы, для чего надо щелкнуть мышью на полосе слева от строки кода.

При запуске программы командой **Отладка, Начать отладку** (**F5**) выполнение программы прервется в точке останова. Далее можно выполнять программу пошагово командами **F10** или **F11**, или продолжить выполнения программы командой **F5** до ее конца или до следующей точки останова.