PROGRAMMING

USING
C/C++

Модуль 1.

Введение в программирование

Что такое программирование?

Программирование *– это процесс разработки и поддержки программного обеспечения.*

В узком смысле программированием называют *процесс написания программы на конкретном языке программирования*. (Этап кодирования)

Этапы разработки ПО

- **Анализ задачи** – этап формализации задачи и определения методов её решения.
- **Проектирование** – этап разработки структуры системы, определения способов взаимодействия компонентов, выбора структур для хранения данных, выбор алгоритма и т.д.
- **Кодирование** – этап написания программного кода компонентов системы.
- **Тестирование** – этап исследования и испытания программного продукта.
- **Документирование** – этап составления руководств и документации к программному продукту.
- **Сопровождение** – этап улучшения, оптимизации ПО после передачи в эксплуатацию.

Что такое программа?

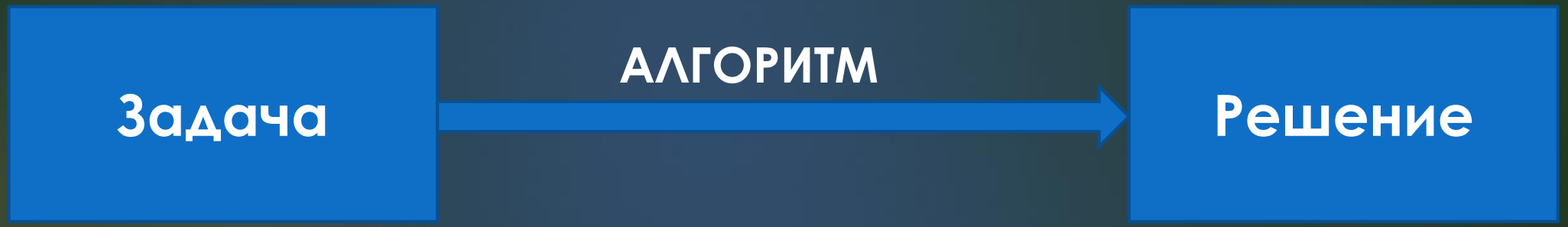
- **Программа** — представленная в объективной форме совокупность данных и команд, предназначенных для выполнения ЭВМ с целью получения определённого результата.

- **Данные** – это информация, которую обрабатывает программа.

Алгоритм

Алгоритм – это точный набор инструкций, описывающий порядок действий исполнителя для достижения цели – решения задачи.

Алгоритм



Алгоритм

Свойства алгоритма:

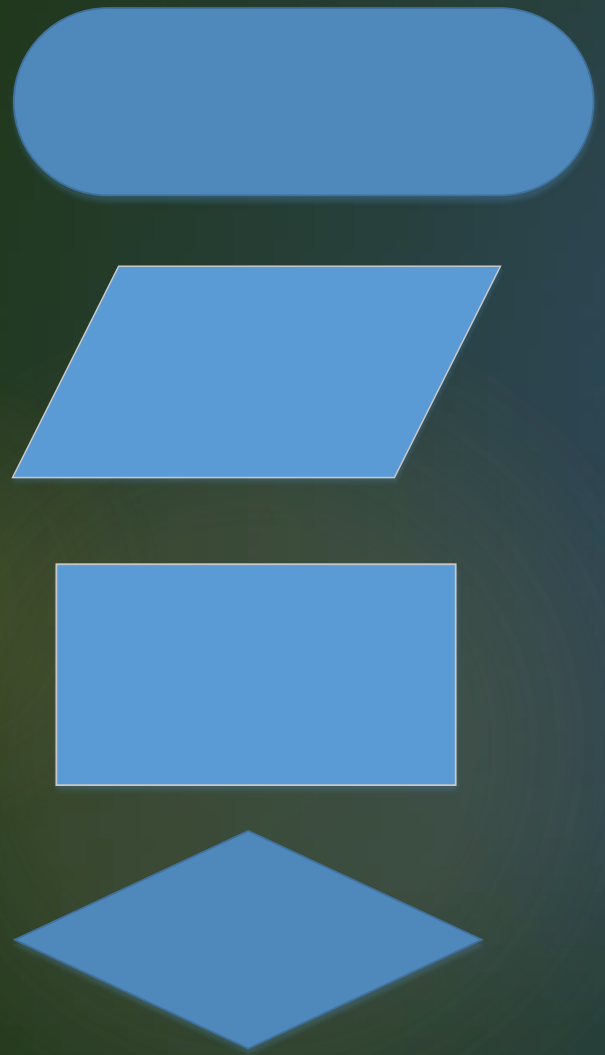
- **Понятность** – алгоритм должен включать только те инструкции, которые доступны и понятны исполнителю.
- **Определенность** – в любой момент времени следующий шаг работы однозначно определяется состоянием системы.
- **Конечность** – алгоритм должен состоять из конечной последовательности шагов.
- **Результативность** – алгоритм должен завершиться с определенным результатом – решением поставленной задачи.
- **Массовость** – алгоритм может применяться к некоторому набору входных данных.
- **Эффективность** – алгоритм должен минимально расходовать ресурсы.

Способы записи алгоритмов

Блок-схема –

это схематическая запись алгоритма, в виде последовательности блоков и соединительных линий, связывающих их.

Блок-схемы



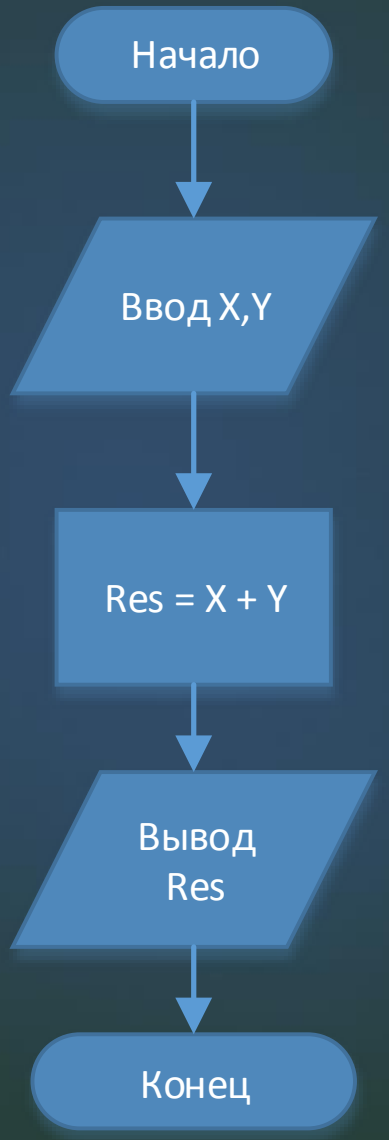
- **Начало/Конец алгоритма**
- **Ввод/вывод данных**
- **Блок действия**
- **Блок условия**

Структура алгоритма

- Линейная – инструкции выполняются последовательно друг за другом.
- Разветвленная – алгоритм, который содержит условие, в результате проверки которого может произойти разделение на несколько ветвей алгоритма.
- Циклическая – алгоритм, который предусматривает многократное повторение одних и тех же инструкций.

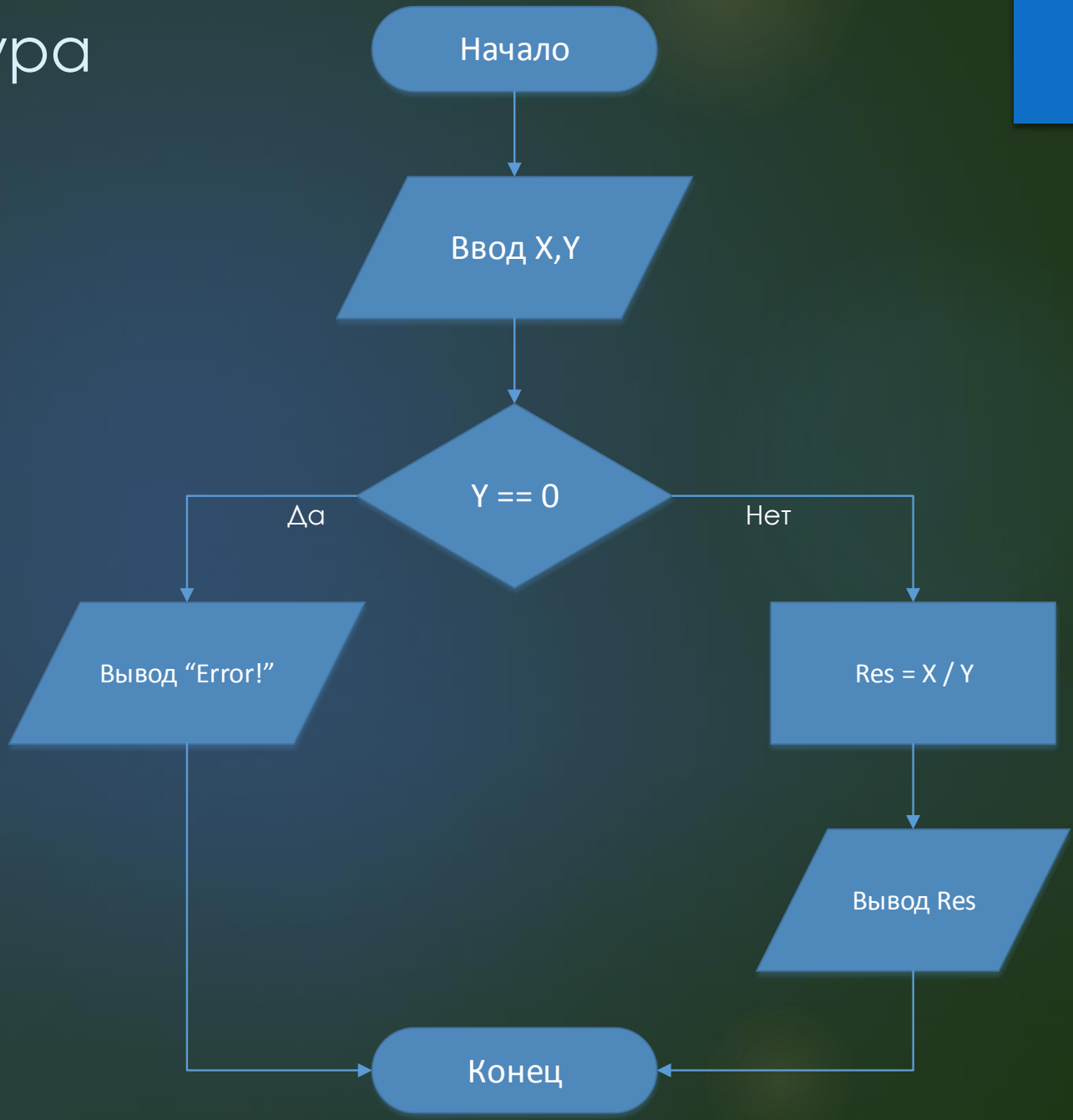
Линейная структура

На примере алгоритма суммирования двух чисел



Разветвленная структура

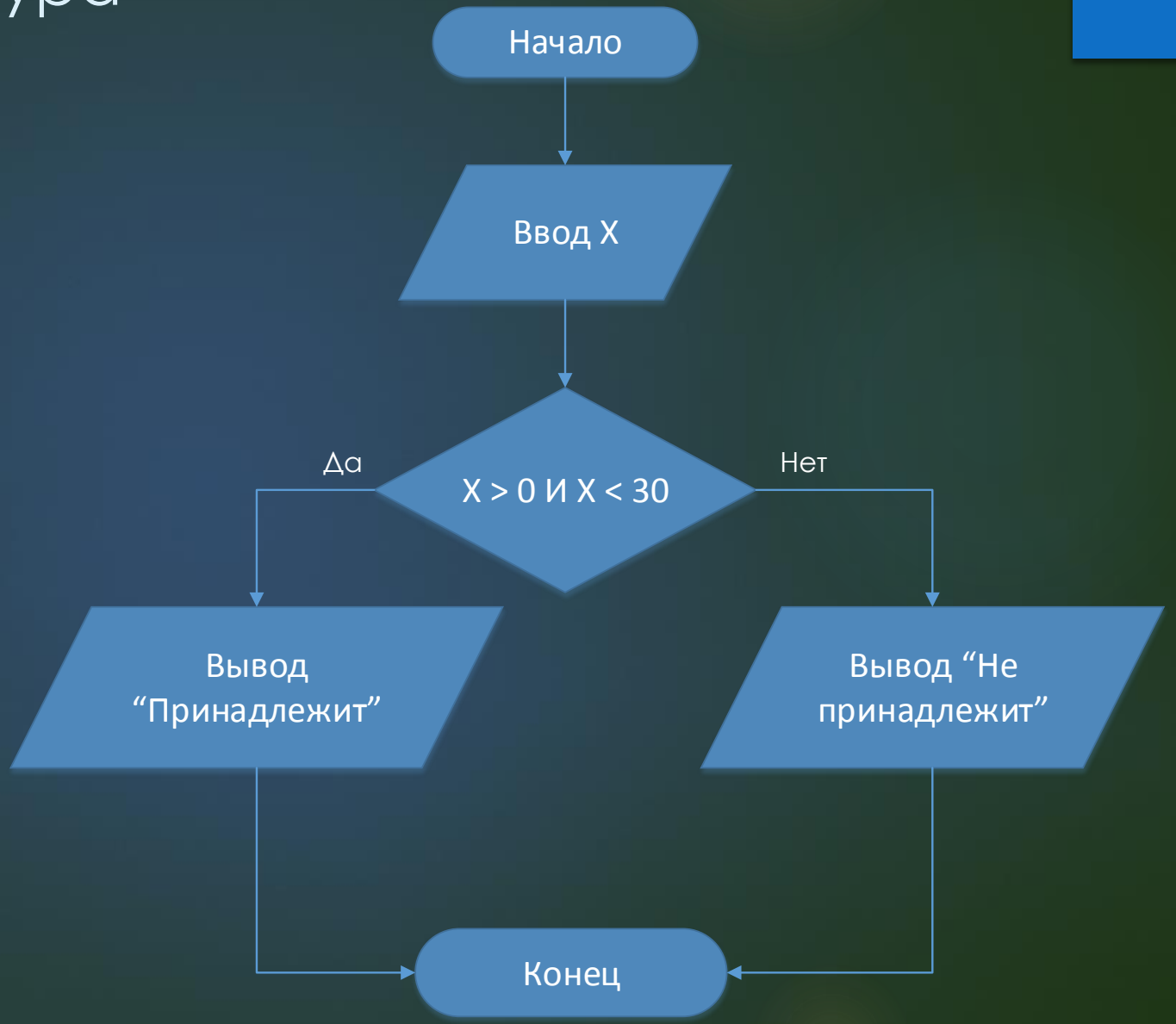
На примере алгоритма деления двух чисел



<u>Обратите внимание</u>, что блок конца у алгоритма только **один!**

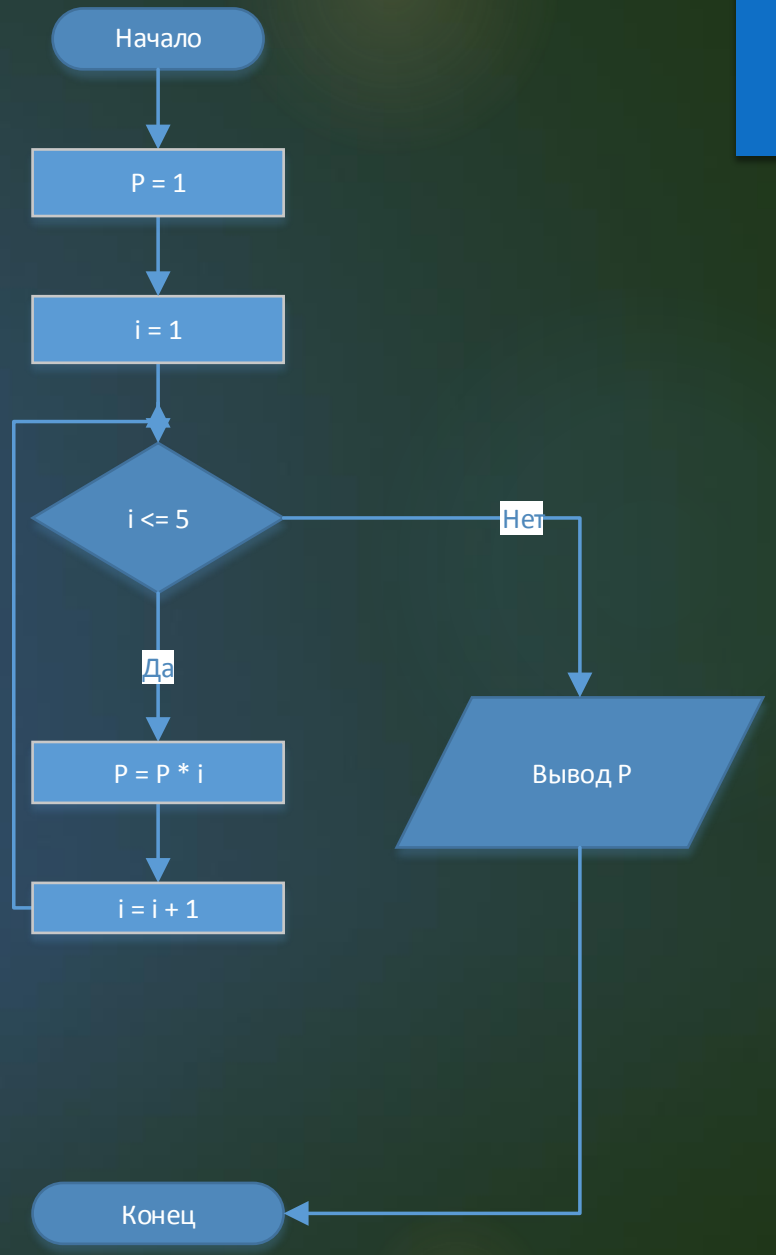
Разветвленная структура

Пример: Проверить принадлежит число диапазону от 0 до 30



Циклическая структура

Пример: Алгоритм вычисления произведения всех чисел от 1 до 5

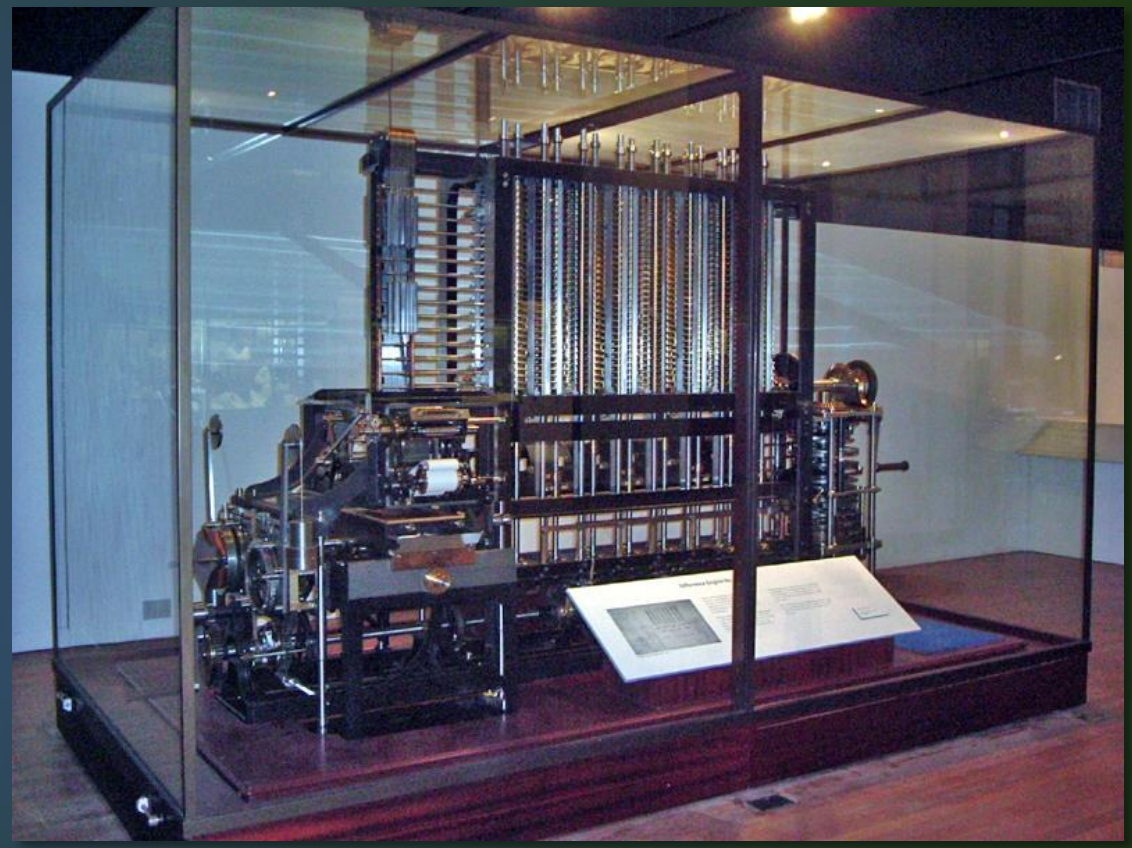


В данном примере присутствуют 2 блока действий, которые повторяются.

Повторение происходит до тех пор, пока условие не станет ложным.

История программирования

Разностная машина Бэббиджа



История программирования

Машинный код

BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9
CD 20 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21

Программа (в шестнадцатеричном представлении),
выводящая на экран текст «**Hello, World!**»

История программирования

Код на языке ассемблера

Assembler является языком **низкого уровня**

В языке ассеблера команды процессору представлены в виде мнемонических последовательностей.

Адрес	Команды		Комментарии
Вызывающая программа			
	⋮		
2000	PUSH	PARAM2	Помещение параметров в стек
2006	PUSH	PARAM1	
2012	CALL	SUB1	
2017	POP	RESULT	Сохранение результата
	ADD	ESP,4	Восстановление уровня стека
	⋮		
Первая подпрограмма			
2100 SUB1	PUSH	EBR	Сохранение регистра указателя на фрейм
	MOV	EBR,ESP	Загрузка указателя на фрейм
	PUSH	EAX	Сохранение регистров
	PUSH	EBX	
	PUSH	ECX	
	PUSH	EDX	
	MOV	EAX,[EBR+8]	Загрузка первого параметра
	MOV	EBX,[EBR+12]	Загрузка второго параметра
	⋮		
	PUSH	PARAM3	Помещение параметра в стек
2160	CALL	SUB2	
2165	POP	ECX	Выталкивание результата работы SUB2 в ECX
	⋮		
	MOV	[EBR+8],EDX	Помещение результата в стек
	POP	EDX	Восстановление регистров
	POP	ECX	
	POP	EBX	
	POP	EAX	
	POP	EBP	Восстановление регистра указателя на фрейм
	RET		Возврат в главную программу
Вторая подпрограмма			
3000 SUB2:	PUSH	EBR	Сохранение регистра указателя на фрейм
	MOV	EBP,ESP	
	PUSH	EAX	Загрузка указателя на фрейм
	PUSH	EBX	Сохранение регистров
	MOV	EAX,[EBP+8]	Загрузка параметров
	⋮		
	MOV	[EBP+8],EBX	Помещение результата SUB2 в стек
	POP	EBX	Восстановление регистров
	POP	EAX	
	POP	EBP	Восстановление регистра указателя на фрейм
	RET		Возврат в первую подпрограмму

Языки высокого и низкого уровня

Чем выше уровень языка, тем больше мы **абстрагируемся** от деталей устройства машины.

Языки высокого и низкого уровня

Сравнение

Языки **высокого** уровня	Языки **низкого** уровня
Нет прямого доступа к аппаратным ресурсам	Прямой доступ к аппаратным ресурсам
Скорость выполнения программы ниже	Высочайшая скорость выполнения программы
Размер исполняемого файла больше	Минимальный размер исполняемого файла
Возможно неоптимальное использование платформы	Максимальное использование специфики платформы
Высокая скорость и низкая сложность разработки	Огромная сложность разработки

Языки высокого и низкого уровня

Сравнение

Основной недостаток языков *низкого* уровня:

- Высочайшая сложность программирования и огромные размеры исходного кода

Основное преимущество языков *высокого* уровня:

- Высокая скорость разработки сложных программных проектов

Трансляция программ

Для выполнения программы на компьютере её необходимо **транслировать** – перевести в машинный код.

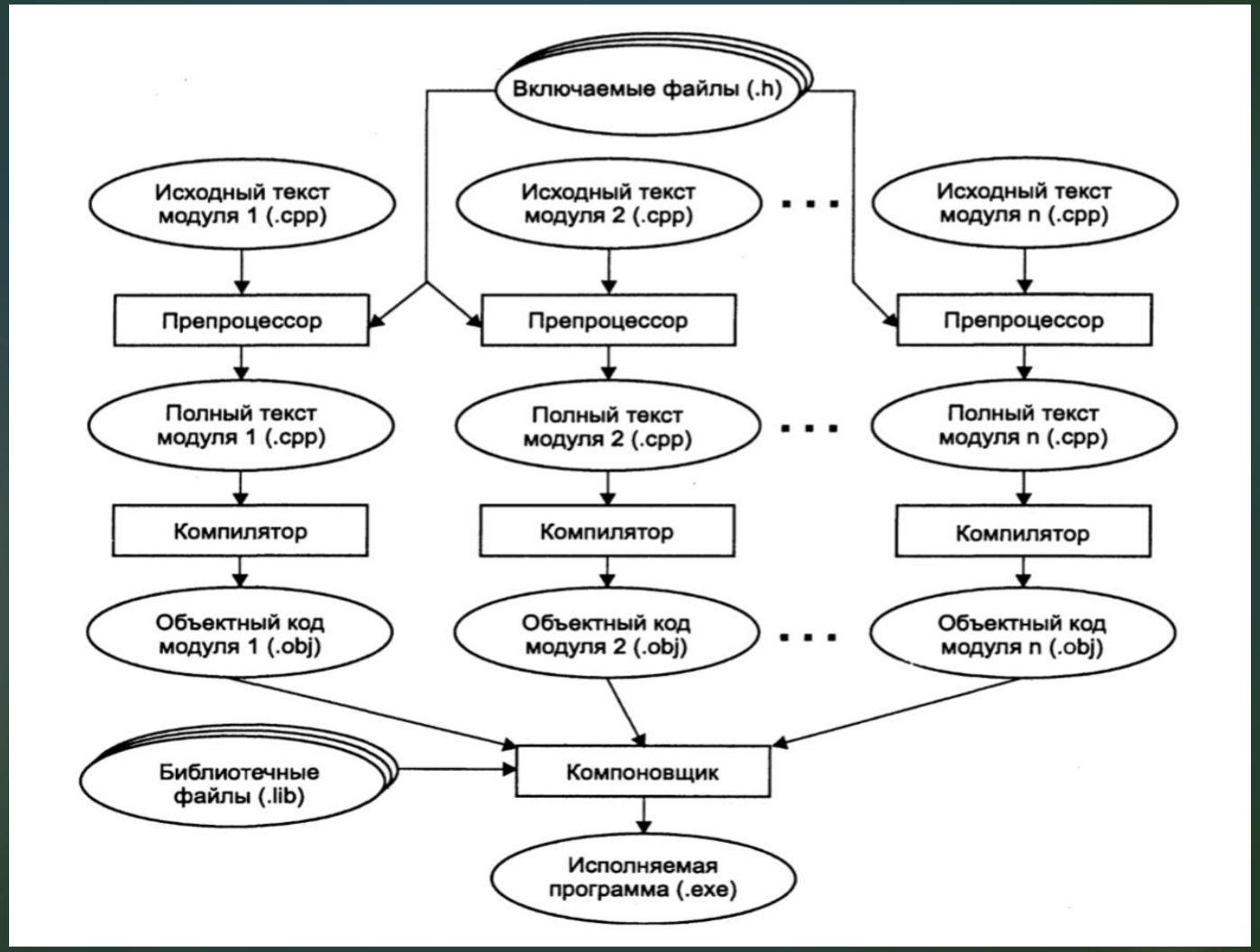
Существует 2 метода – трансляции:

- **Компиляция**
  - Исходный код программы переводится с определенного языка программирования на *машинный код* специальной программой – **компилятором**.
  - Машинный код может **многократно** выполняться процессором.
  - Машинный код является **непереносимым** между разными *аппаратными и программными платформами*.

- **Интерпретация**
  - Программа, команда за командой, переводится в машинный код, и каждая инструкция **сразу же** выполняется специальной программой – **интерпретатором**.
  - На компьютере, который запускает такую программу, ***обязательно должен быть установлен*** интерпретатор.
  - Это позволяет добиться переносимости между платформами.

Этапы сборки программы

Создание исполняемого файла (.exe)



Препроцессор

Препроцессор – это специальная программа, которая обрабатывает исходный код до компиляции.

Препроцессор подключает заголовочные файлы (.h), удаляет лишние пробелы, комментарии и т.д.

Компилятор

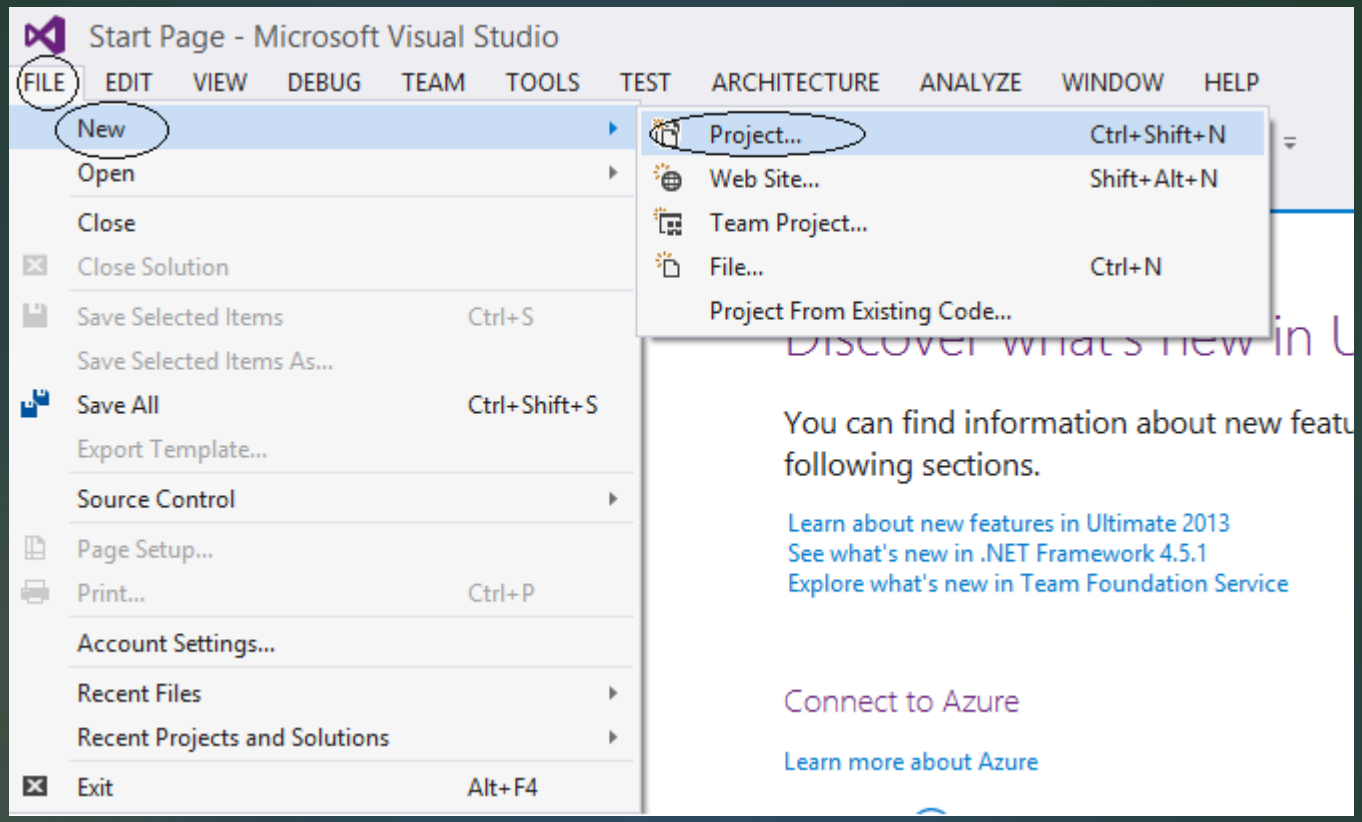
Компилятор – проверяет исходный код на синтаксические ошибки, если их нет – транслирует *исходный код* в *машинный код*.

На выходе получается **объектный модуль (.obj)**

Компоновщик (linker)

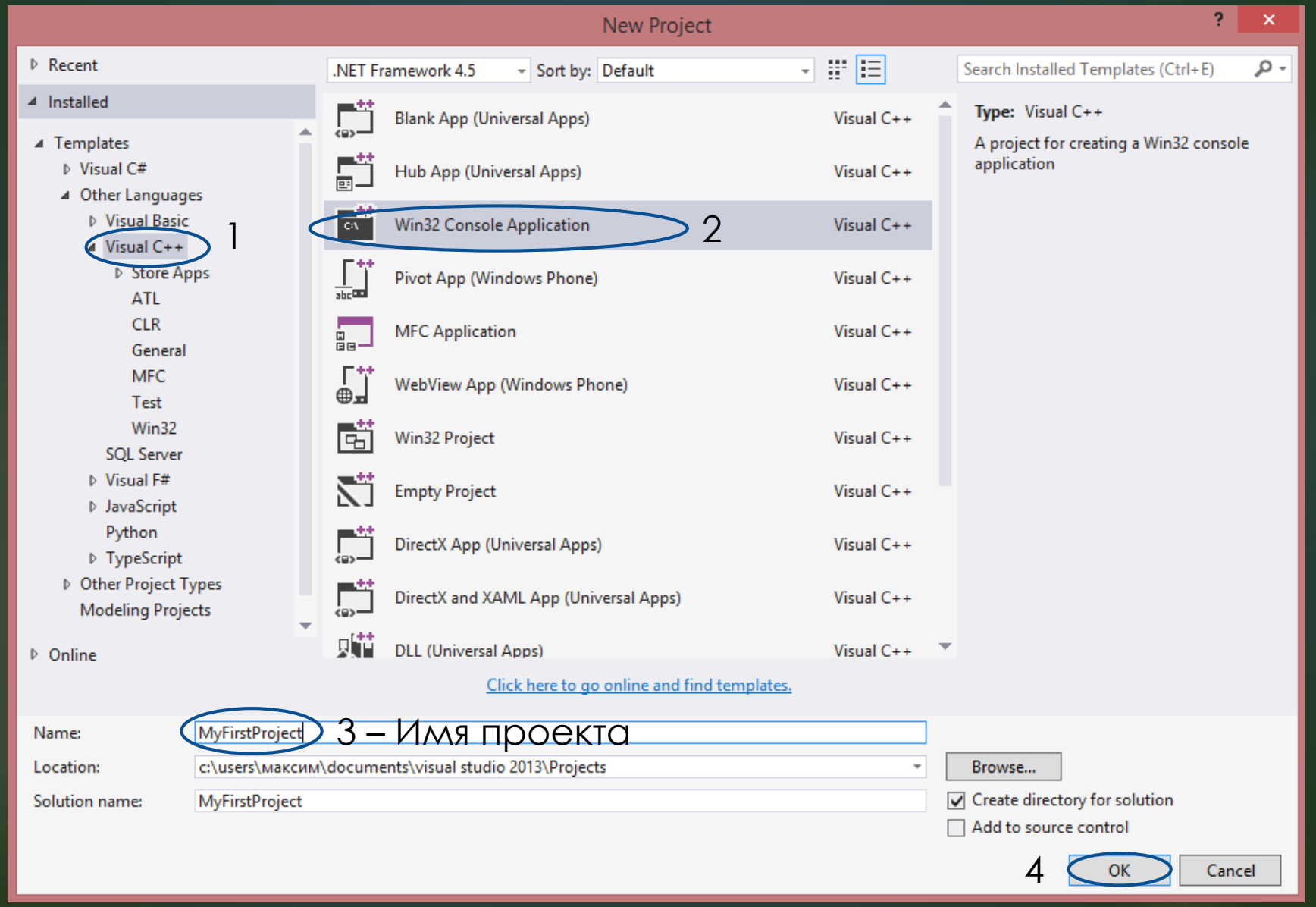
Компоновщик – формирует исполняемый модуль программы из **объектных модулей** и **библиотечных файлов** (.lib)

Создание проекта в среде Visual Studio 2013

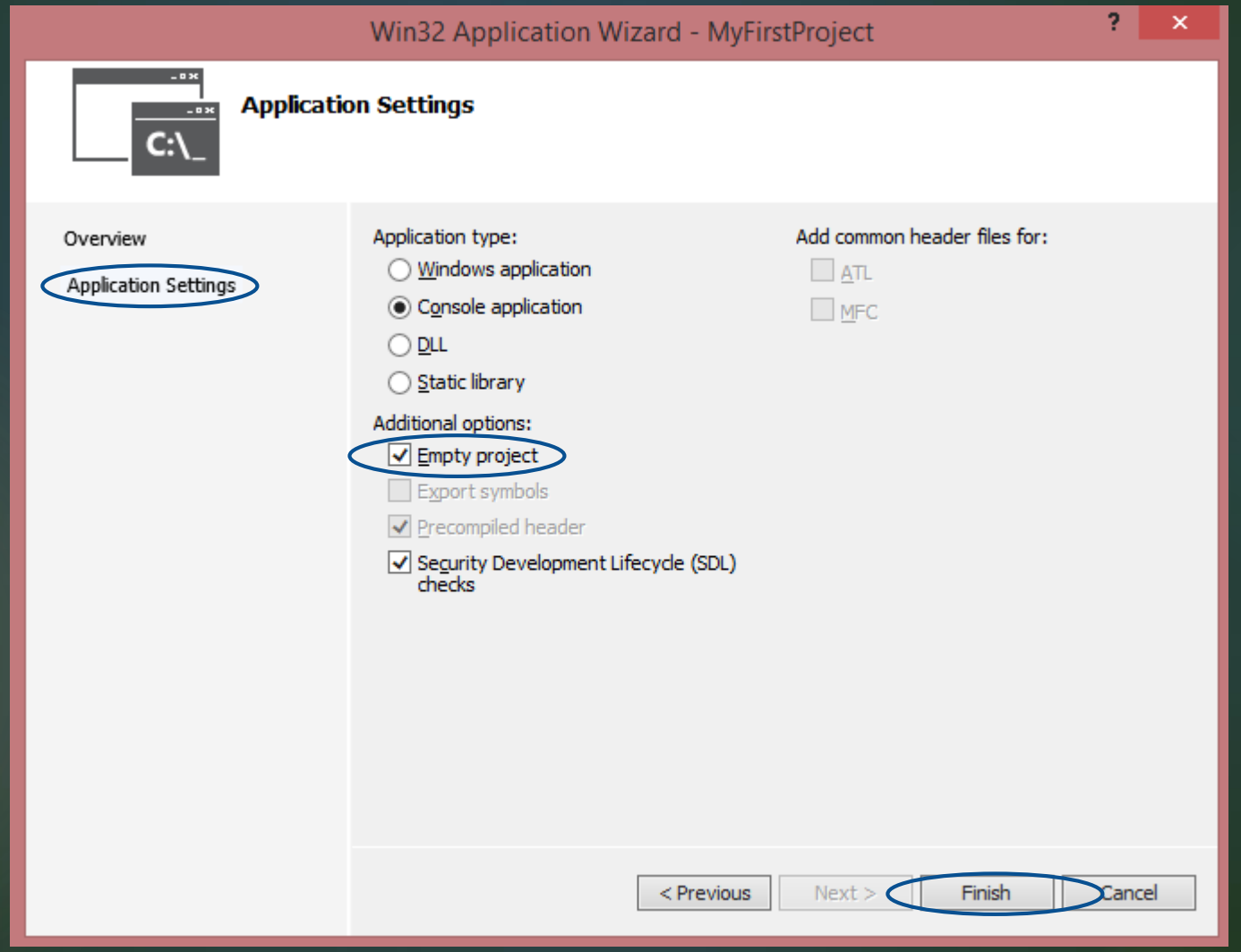


Пункт меню **File -> New -> Project**

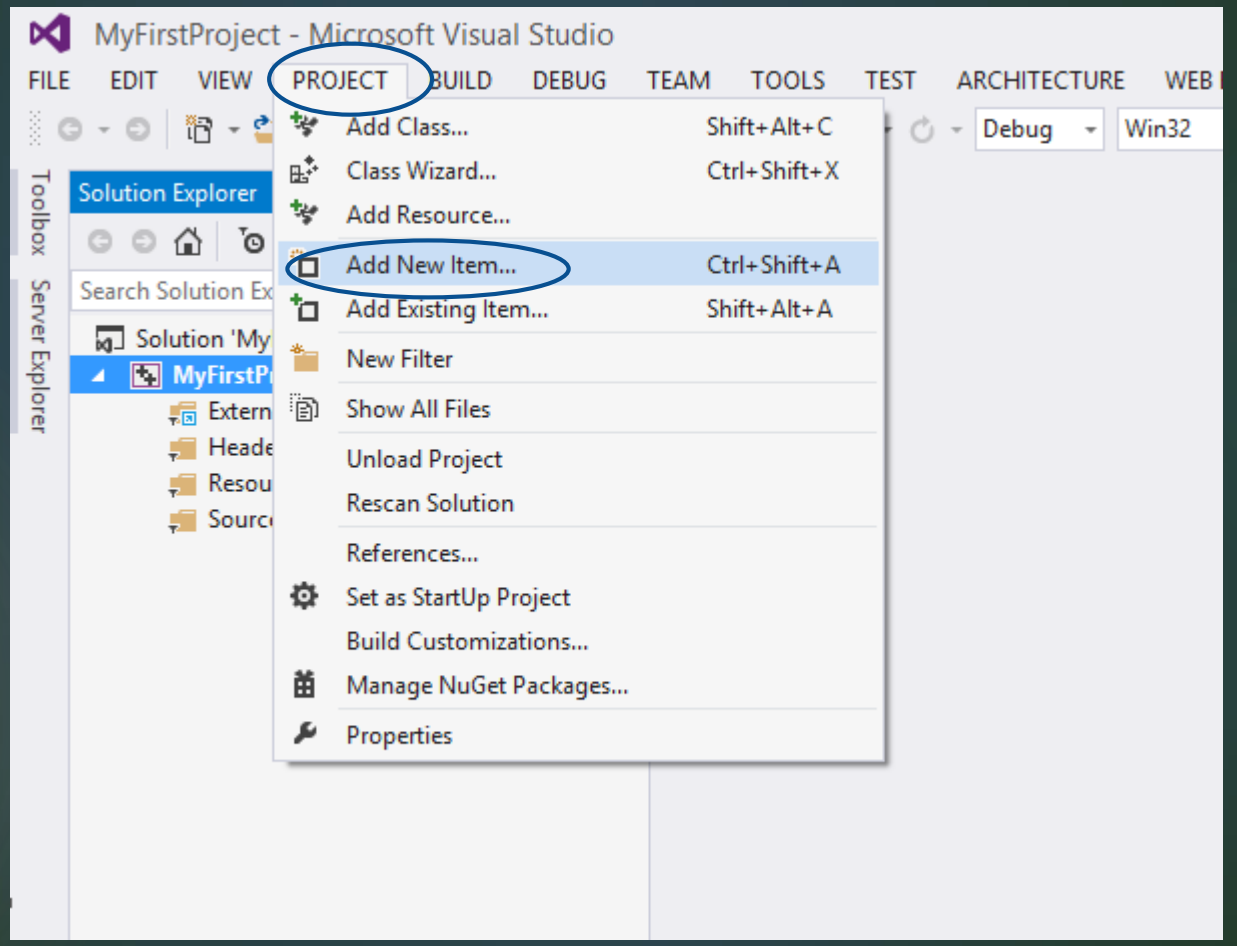
Создание проекта в среде Visual Studio 2013



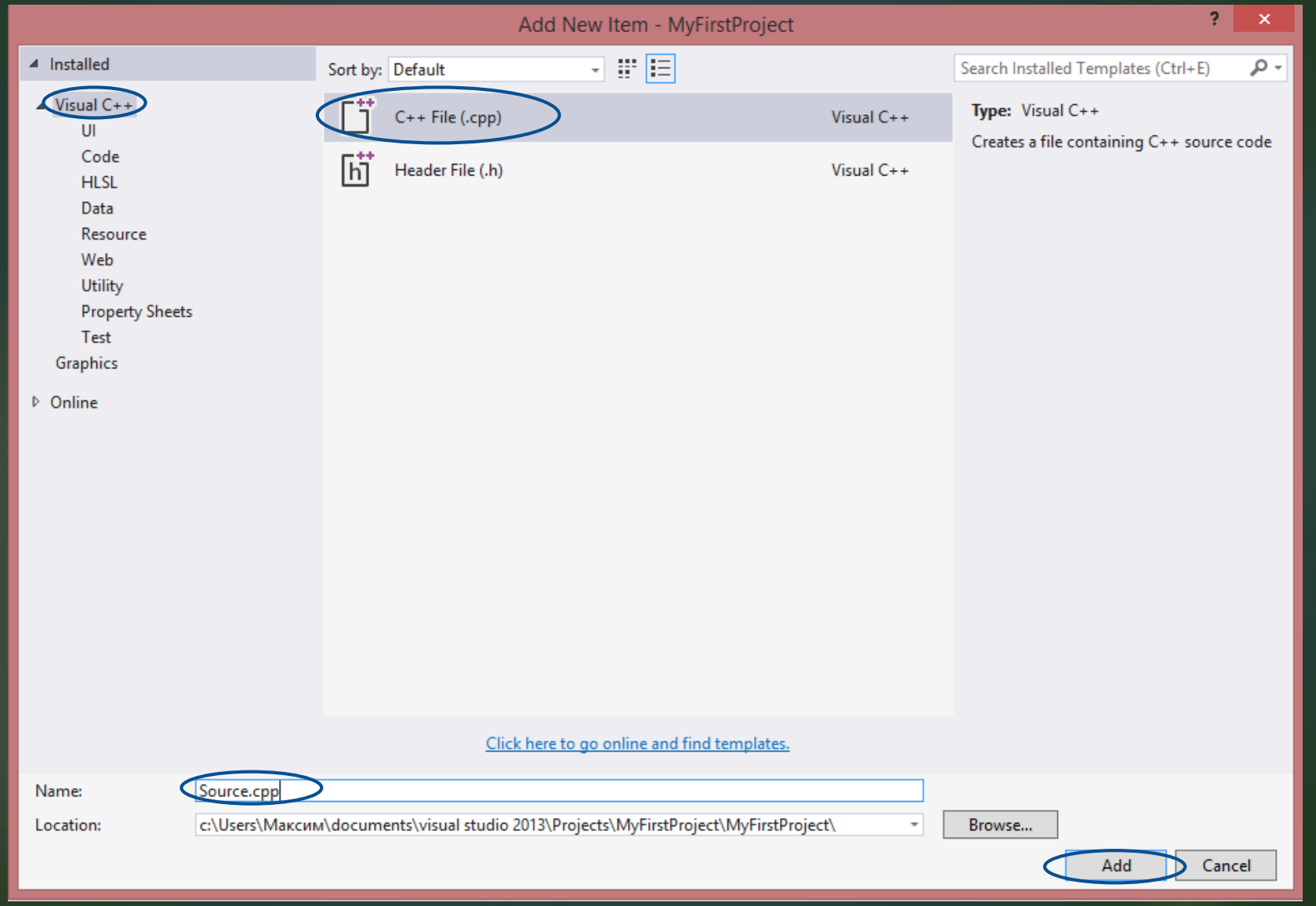
Создание проекта в среде Visual Studio 2013



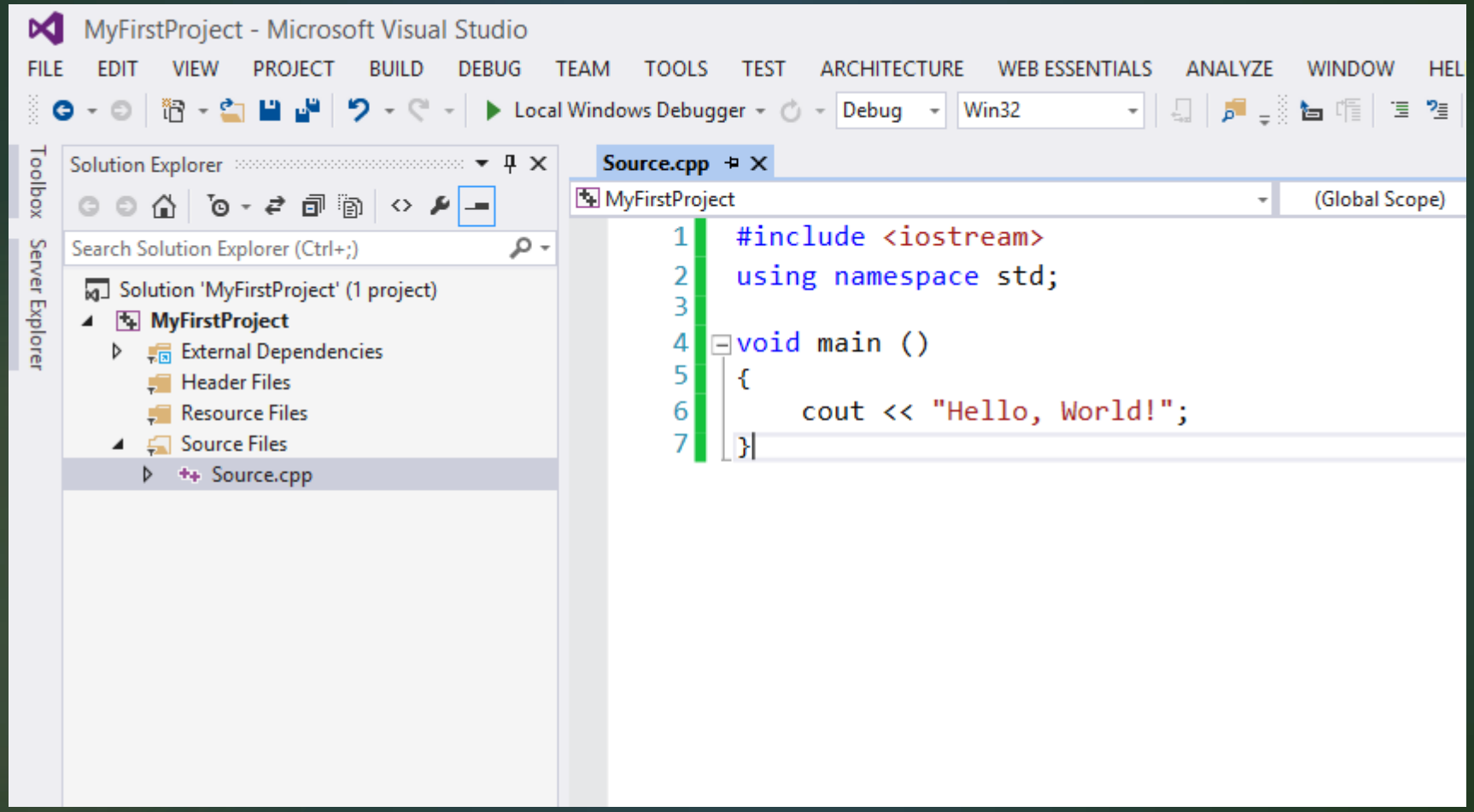
Создание проекта в среде Visual Studio 2013



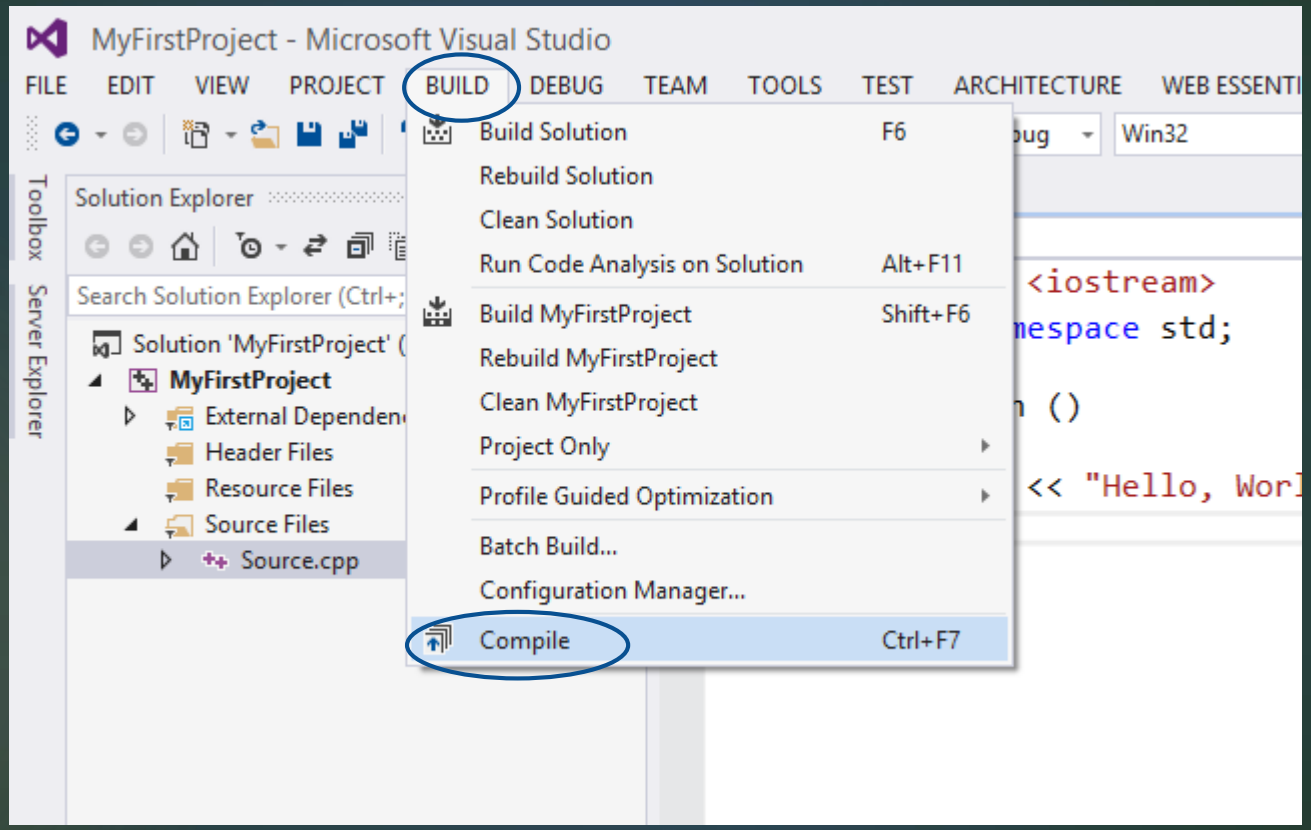
Создание проекта в среде Visual Studio 2013



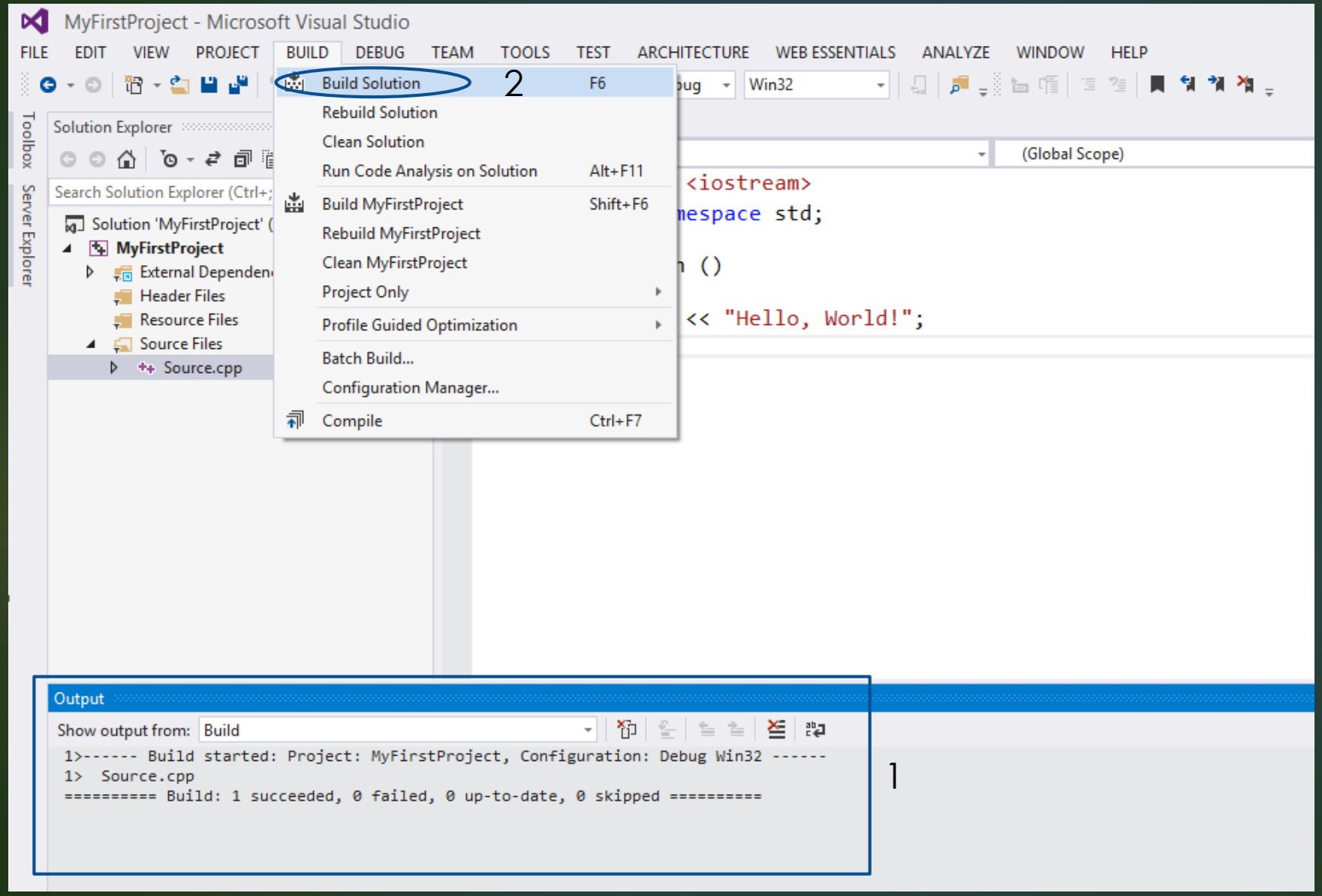
Создание проекта в среде Visual Studio 2013



Создание проекта в среде Visual Studio 2013



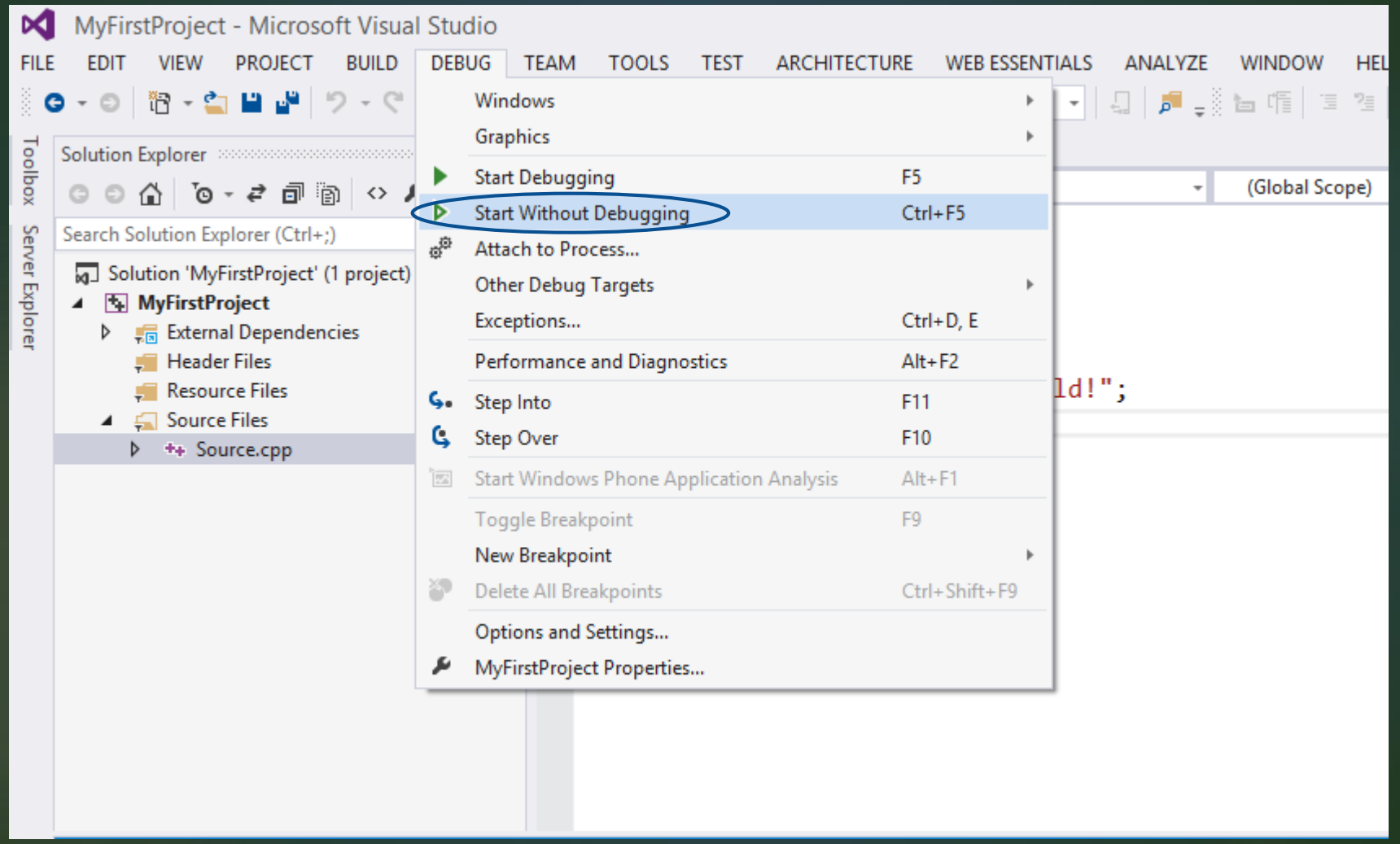
Создание проекта в среде Visual Studio 2013



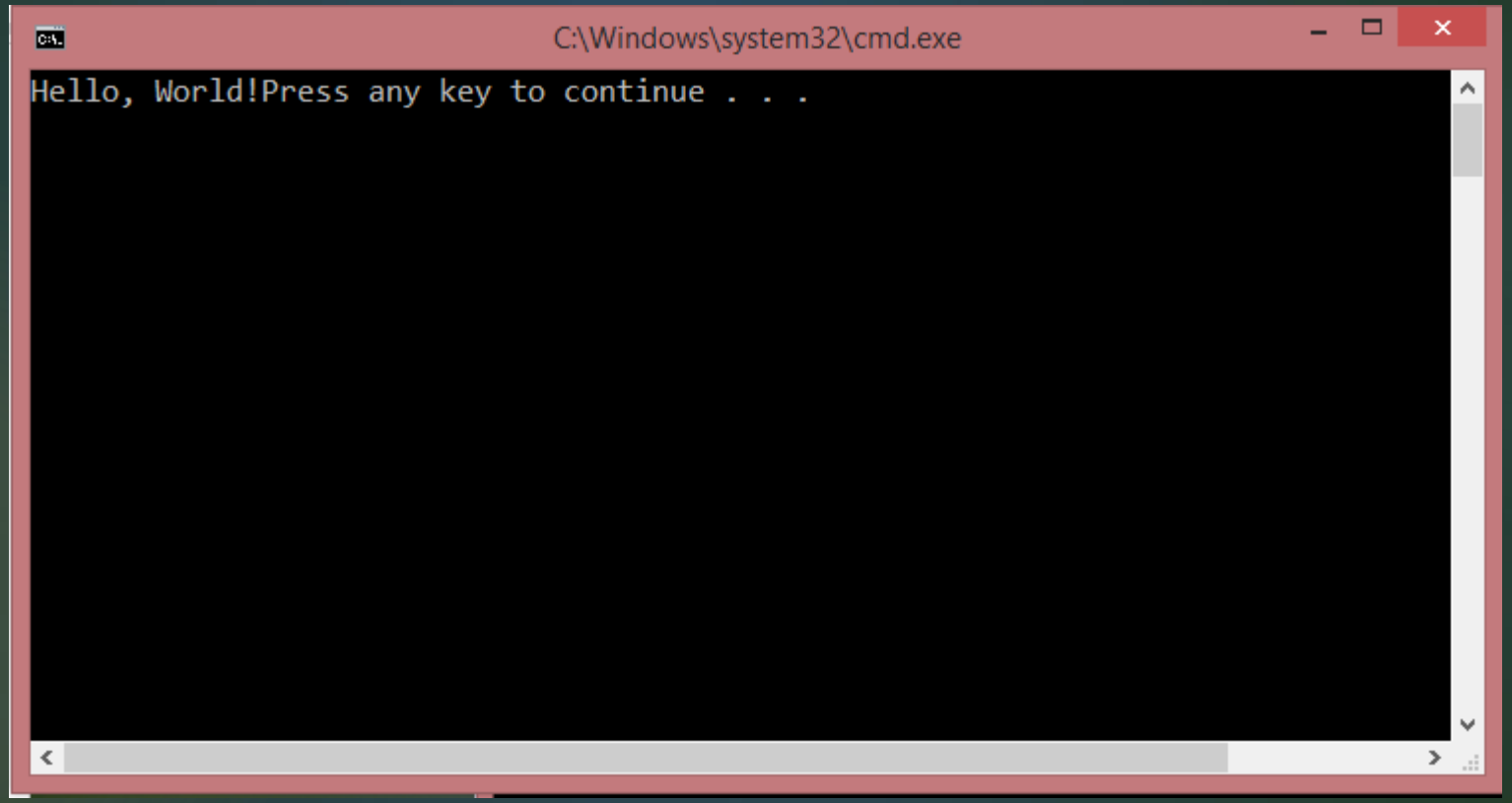
1. В окне **Output** выводится информация об ошибках и предупреждениях, которые возникают в процессе компиляции. Если их нет – можно выполнять компоновку (сборку) проекта.

2. Выполняем пункт меню **Build Solution**. Это запустит компоновщик.

Создание проекта в среде Visual Studio 2013



Создание проекта в среде Visual Studio 2013



Команда cout

Команда cout используется для вывода данных на экран.

Синтаксис:

```
cout    <<      “Text”;
```

cout позволяет выводить сразу несколько порций данных в одной инструкции

```
cout  <<   “Text” << “New Text” << “New Text 2”;
```

Escape-последовательности

Символы, которые имеют специальное назначение могут быть представлены с помощью так называемых **Escape-последовательностей (управляющих последовательностей).**

Escape-последовательность – это специальная совокупность символов, которая начинается с символа обратного слэша (\ ) и заменяется транслятором на определенный символ, который:

1. Либо имеет специальное значение
2. Либо его нельзя использовать в данном контексте
3. Либо он отсутствует на клавиатуре

Escape-последовательности

Esc. последовательность	Значение (Hex)	Значение (Dec)	ASCII обозначение	Назначение
\0	**0x00**	**0**	**NUL**	**Нулевой символ**
\a	0x07	7	BEL	Звонок, beep
\b	0x08	8	BS	Шаг назад, backspace
\f	0x0C	12	FF	Прогон страницы
\n	**0x0A**	**10**	**LF**	**Перенос строки**
\r	0x0D	13	CR	Возврат каретки
\t	**0x09**	**9**	**HT**	**Горизонтальная табуляция**
\v	0x0B	11	VT	Вертикальная табуляция
****	**0x5C**	**92**	****	**Обратный слэш**
\'	**0x2C**	**44**	**‘**	**Апостроф**
\"	**0x22**		**“**	**Кавычки**
\?	0x0F	15	?	Вопросительный знак
\ddd				**Вывод символа по коду (от 1 до 3-х восьмеричных цифр)**
\xhh	**0xhh**			**Вывод символа по коду (шестнадцатиричные цифры)**