Лекция 1: Введение в мир программирования. От идеи к программе.

Что такое программирование?

Если дать сухое определение, то **программирование** — это процесс создания компьютерных программ.

Но давайте посмотрим глубже. Это искусство и наука преобразования идеи в четкую, однозначную последовательность инструкций, понятных машине. Это способность автоматизировать рутину, решать сложные задачи и создавать нечто совершенно новое: от простого калькулятора до сложнейшей нейросети, от сайта вашей любимой социальной сети до игры, в которую вы играете.

> *«Программирование — это разбиение чего-то большого и невозможного на что-то маленькое и вполне реальное.» — **Jazzwant**.*

---

1. Немного истории: от ткацких станков до кремния

Чтобы понять, где мы находимся, давайте оглянемся назад.

- **XIX век:** Ада Лавлейс, дочь поэта Байрона, написала первую в мире программу для аналитической машины Чарльза Бэббиджа. Её по праву называют **первым программистом**. Она поняла, что машина может делать не только вычисления, но и работать с символами, предвосхитив будущее компьютеров.
- **1940-е годы:** Появление первых электронно-вычислительных машин (ЭВМ). Программирование тогда было физическим трудом — программы вводились с помощью переключения тумблеров и перфокарт.
- **1950-е:** Появление первых языков программирования высокого уровня, таких как **Fortran** (для научных расчетов) и **COBOL** (для бизнеса). Они позволили писать команды, больше похожие на человеческий язык, а не на последовательности нулей и единиц (машинный код).
- **1970-е:** Рождение языка **C**. Он стал золотым стандартом системного программирования и оказал огромное влияние на многие последующие языки, включая наш C#.
- **1980-1990-е:** Эра персональных компьютеров. Появление объектно-ориентированного программирования (ООП). Зарождаются языки вроде C++ и Java.
- **2000-е:** Компания Microsoft представляет **C#** (произносится «си-шарп»). Этот язык был создан как современный, универсальный и мощный инструмент для разработки под платформу .NET. Сейчас на C# пишут игры (Unity), десктопные приложения, веб-сервисы, мобильные придания и даже облачные решения.

---

2. Сердце программирования: Алгоритм

Прежде чем писать код, нужно понять *что* писать. Здесь на сцену выходит **алгоритм**.

Алгоритм — это четкая, конечная последовательность шагов, решающая конкретную задачу.

Пример из жизни: «Приготовить чай»

1. Взять чашку.

2. Положить в неё чайный пакетик.
3. Вскипятить воду.
4. Залить пакетик кипятком.
5. Подождать 3 минуты.
6. Вынуть пакетик.
7. (По желанию) Добавить сахар или лимон.

Всё! Это и есть алгоритм. Если вы будете следовать ему шаг за шагом, вы получите результат.

> *«Не волнуйтесь, если что-то не работает. Если бы всё работало, вас бы уволили.» — **Закон программирования Мошерта**.*

В программировании то же самое. Прежде чем писать код для калькулятора, вы должны продумать алгоритм: "Пользователь вводит два числа и операцию. Если операция '+', то сложить числа. Если '-', то вычесть, и т.д. Показать результат."

Свойства хорошего алгоритма:

- **Понятность:** Каждый шаг должен быть ясным.
- **Дискретность:** Процесс разбит на отдельные, мелкие шаги.
- **Определенность:** Каждое действие однозначно.
- **Результативность:** Алгоритм должен всегда приводить к результату.
- **Массовость:** Один и тот же алгоритм должен решать класс похожих задач (не только для чисел 5 и 3, а для любых чисел).

---

3. Стили программирования: Парадигмы

Как можно подходить к решению одной и той же задачи? По-разному. Эти подходы называются **парадигмами программирования**.

- **Императивное программирование:** Программа как список команд, которые изменяют состояние программы. "Сделай это, потом сделай то".
- **Процедурное программирование:** Разновидность императивного. Код организуется в процедуры (функции, методы) — блоки кода, решающие подзадачу. Например, функция `СложитьДваЧисла()`.
- **Объектно-ориентированное программирование (ООП):** Это одна из самых важных и популярных парадигм, и C# построен вокруг неё.

ООП — это модель, где программа представляется в виде набора взаимодействующих объектов.

У каждого объекта есть:

- **Данные (Поля/Свойства):** *Что объект "знает" о себе?* (например, у объекта `Студент` есть `Имя`, `Фамилия`, `СреднийБалл`).
- **Поведение (Методы):** *Что объект "умеет делать"?* (например, `Студент.СдатьЭкзамен()`, `Студент.ПосетитьЛекцию()`).

Основные принципы ООП:

1. **Инкапсуляция:** Объединение данных и методов в одном объекте + сокрытие внутренней реализации. Мы пользуемся телефонам, не зная, как именно работает каждый транзистор внутри.

2. **Наследование:** Возможность создать новый класс на основе существующего, заимствуя его свойства и поведение. Класс `Аспирант` может наследоваться от класса `Студент`, добавляя новое поле `ТемаДиссертации`.
3. **Полиморфизм:** Объекты разных классов могут реагировать на одно и то же действие по-разному. Метод `Нарисовать()` у объектов `Круг`, `Квадрат` и `Треугольник` будет работать по-разному, хотя название одно.

- **Функциональное программирование:** Программа как вычисление математических функций, без изменения состояния.

C# поддерживает несколько парадигм, что делает его очень гибким.

---

4. Принципы хорошего кода

Писать код, который работает, — это полдела. Важно писать код, который будут **понимать** другие люди (и вы сами через месяц).

- **KISS (Keep It Simple, Stupid):** Делай проще.
- **DRY (Don't Repeat Yourself):** Не повторяйся. Если один и тот же код встречается дважды, вынеси его в функцию.
- **Читаемость:** Имена переменных и функций должны быть осмысленными. `userAge` вместо `a`, `CalculateTotalPrice()` вместо `func1()`.
- **Комментарии:** Комментируйте сложные или неочевидные моменты в коде.

> *«Любой дурак может написать код, который поймет компьютер. Хорошие программисты пишут код, который поймут люди.» — **Мартин Фаулер**.*

---

5. Как компьютер понимает программу? Этапы сборки

Мы пишем код на языке, понятном человеку (C#), но компьютер понимает только нули и единицы (машинный код). Превращение одного в другое — это процесс сборки.

Этап 1: Написание кода и Препроцессинг

Это этап, где работаете вы и компилятор *до* перевода кода в IL.

1. **Написание кода:** Вы создаете файлы .cs с кодом на C#.
2. **Лексический и синтаксический анализ:** Компилятор читает исходный текст, разбивает его на "слова" (токены) — ключевые слова, идентификаторы, операторы — и проверяет, соответствует ли их последовательность правилам грамматики C#. Если нет, вы получаете синтаксические ошибки (например, пропущенную точку с запятой).
3. **Семантический анализ:** Компилятор проверяет *смысл* кода. Соответствуют ли типы данных? Объявлены ли используемые переменные и методы? На этом этапе вы получаете ошибки вроде "The name 'variableName' does not exist in the current context".
4. **Обработка директив препроцессора (Условная компиляция):** Вот где в игру входят директивы, начинающиеся с #.

Директивы в языке программирования C# (препроцессорные директивы) — это специальные команды, которые препроцессор распознаёт и выполняет до компиляции программы. Они позволяют изменить текст программы, например, заменить некоторые лексемы, вставить текст из другого файла, запретить трансляцию части текста.

Виды

Некоторые виды директив препроцессора в C#:

- **#include** — включает в текст программы содержимое указанного файла. Имеет две формы:

  #include "имя файла" или #include <имя файла>

- **#define** — задаёт макроопределение (макрос) или символическую константу. Заменяет все последующие вхождения идентификатора на текст (макроподстановка).
- **#undef** — отменяет действие директивы #define для указанного идентификатора.
- **#ifdef и #ifndef** — проверяют, определён ли определённый символ или макрос.
- **#else** — может использоваться вместе с #ifdef или #ifndef, чтобы указать, что определённый код должен выполниться, если условие не выполнено.

Синтаксис

Директива (командная строка препроцессора) — строка в исходном коде, имеющая следующий формат:

#ключевое_слово параметры

. В формат входят:

- символ #;
- ноль или более символов пробелов и/или табуляции;
- одно из предопределённых ключевых слов;
- параметры, зависимые от ключевого слова.

Если ключевое слово не указано, директива игнорируется. Если указано несуществующее ключевое слово, выводится сообщение об ошибке и компиляция прерывается.

Примеры использования

- **Включение файлов**. Например:

#include <math.h> — включение объявлений математических функций, #include <stdio.h> — включение объявлений функций ввода-вывода

- **Создание макроса**.

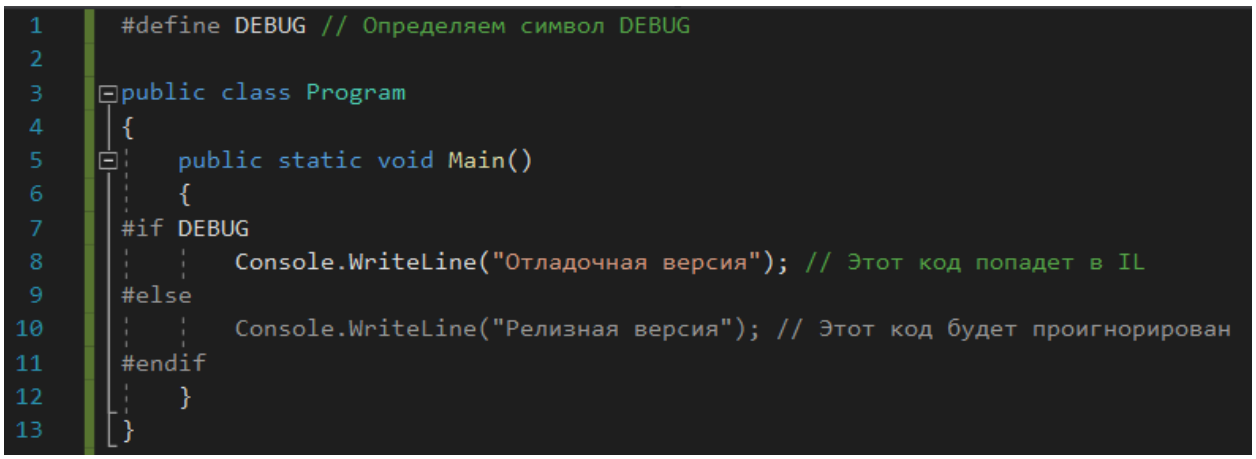
#define MESSAGE "Hello World" — любой экземпляр слова MESSAGE заменяется на текст, указанный в директиве.

- **Проверка, определён ли макрос**.

#ifdef DEBUG — проверяет, определена ли отладка,

#ifndef DEBUG — проверяет, не определена ли отладка.

Пример:



```
#define DEBUG // Определяем символ DEBUG

public class Program
{
    public static void Main()
    {
#if DEBUG
        Console.WriteLine("Отладочная версия"); // Этот код попадет в IL
#else
        Console.WriteLine("Релизная версия"); // Этот код будет проигнорирован
#endif
    }
}
```

Когда это выполняется? Это происходит *во время компиляции*. Компилятор анализирует символы (DEBUG, TRACE и т.д.) и физически вырезает блоки кода, которые не удовлетворяют условию. В промежуточный язык (IL) попадет только активная ветка.

Этап 2: Компиляция в CIL (Common Intermediate Language)

После успешного прохода всех проверок и препроцессинга наступает ключевой этап.

5. **Генерация CIL и метаданных:** Компилятор преобразует ваш код C# в промежуточный язык (CIL, ранее известный как MSIL). Это низкоуровневый, но платформенно-независимый язык, похожий на ассемблер.
    - **CIL** содержит инструкции для виртуальной машины .NET (CLR).
    - **Метаданные** — это подробное описание всего, что есть в вашей сборке: имена классов, методов, их параметры, типы возвращаемых значений, атрибуты и т.д. Метаданные и CIL хранятся вместе в файле сборки (.exe или .dll).

Важный момент: На этом этапе **еще не выполняется ни одна логика вашей программы** (например, вычисления, вызовы методов). Просто создается ее "портрет" на языке CIL.

Этап 3: Сборка в Assembly (Сборку)

6. **Формирование сборки (Assembly):** Скомпилированный CIL-код и метаданные упаковываются в файл, который называется **сборкой**. Это основная единица развертывания в .NET. Сборка может быть исполняемым файлом (.exe) или библиотекой (.dll). Вместе с вашим кодом в сборку включается **манифест**, который содержит информацию о самой сборке (версия, зависимости, культура) и списки всех внешних сборок, которые ей нужны для работы.

Этап 4: Выполнение и JIT-компиляция

Теперь программа собрана, и пользователь ее запускает. Управление переходит к среде выполнения .NET — CLR (Common Language Runtime).

7. **Загрузка сборки:** CLR загружает вашу сборку и анализирует ее манифест, чтобы понять, какие еще сборки (зависимости) нужны. Затем она загружает и эти сборки.
8. **Проверка безопасности и корректности CIL:** (Опционально, зависит от настроек) CLR может проверить CIL-код на типобезопасность (например, не пытается ли программа записать в память, к которой у нее нет доступа).

9. **JIT-компиляция (Just-In-Time):** Это сердце процесса выполнения. JIT-компилятор работает не со всей программой сразу, а по мере необходимости — метод за методом.
    - Когда программа впервые пытается выполнить какой-либо метод (например, Main), JIT-компилятор вступает в игру.
    - Он берет CIL-код *этого конкретного метода* и транслирует его в настоящий **машинный код**, специфичный для процессора и операционной системы, на которой запущено приложение (x86, x64, ARM и т.д.).
    - При этом JIT-компилятор проводит финальные, самые агрессивные оптимизации, так как знает точные характеристики текущего "железа".
10. **Выполнение машинного кода:** Скомпилированный JIT-компилятором машинный код сохраняется в памяти и выполняется процессором напрямую. При последующих вызовах этого же метода CLR уже использует готовый машинный код, минуя этап JIT-компиляции.

Сводная таблица: "Что и когда выполняется?"

Этап	Что происходит?	Когда выполняется?	"Исполнитель"
Препроцессинг	Обработка #if, #define. Условное исключение кода.	Во время компиляции	Компилятор C# (csc.exe, Roslyn)
Компиляция	Проверка синтаксиса, типов. Генерация CIL и метаданных.	Во время компиляции	Компилятор C#
Построение сборки	Упаковка CIL в .exe/.dll файл (сборку).	Во время компиляции	Компилятор C#
JIT-компиляция	Перевод CIL в машинный код (по методам).	**Во время выполнения программы**, при первом вызове метода	JIT-компилятор (.NET CLR)
Исполнение	Непосредственное выполнение инструкций процессором.	**Во время выполнения программы**, после JIT-компиляции метода	Процессор компьютера

Итог

Эта двухэтапная система (C# -> CIL -> Машинный код) — это основа кроссплатформенности .NET. Одна и та же сборка с CIL может быть запущена на Windows, Linux или macOS, потому что у каждой из этих платформ есть своя реализация среды выполнения (CLR) со своим JIT-компилятором, который уже знает, как превратить CIL в нужный машинный код.

---

6. Что впереди?

В этом курсе будет пройден путь от самых основ — переменных, типов данных, условных операторов `if` и циклов `for` — до понимания принципов ООП и создания первых настоящих приложений.