1 вводная лекция.

План:

1. История языков программирования и процесса разработки ПО с начала (1950-е) до наших дней. Машинный код + перфокарты, языки ассемблера, ForTran/ALGOL, LISP, PASCAL, COBOL, C, SmallTalk, C++, Basic.
2. Подходы в разработке ПО. Спагетти-код, структурный подход, ООП, функциональщина.
3. Кроссплатформенность, переносимость, компиляция VS интерпретация и прочие причины возникновения Java.
4. Внезапно – C#. Следующий виток эволюции.
5. C# и .NET Framework

2 вводная лекция.

1. Какие платформы бывают.
2. Какие задачи встречаются.
3. Какие профессии нужны в IT?
4. Что нужно знать для становления junior .NET developer? И сколько времени это займет.
5. О чем мы расскажем на курсах – план занятий по шарпу.
6. Обзор других технологий и зарплат.



1 вводная лекция.

**История языков программирования и процесса разработки ПО с начала (1950-е) до наших дней. Машинный код + перфокарты, языки ассемблера, ForTran/ALGOL, LISP, PASCAL, COBOL, C, SmallTalk, C++, Basic.**

Для управления первыми вычислительными машинами использовались перфокарты. Перфока́рта — [носитель информации](https://dic.academic.ru/dic.nsf/ruwiki/4843), предназначенный для использования системами автоматической обработки данных. Сделанная из тонкого [картона](https://dic.academic.ru/dic.nsf/ruwiki/133890), перфокарта представляет информацию наличием или отсутствием отверстий в определённых позициях карты (могу принести из института, Давыдкина на них же пишет примеры). Информация, которую они содержат, и есть программа – комбинации отверстий или их отсутствия, которые обозначают определённые действия, которые машина должна выполнить.

С развитием компьютерной техники появился [машинный язык](https://dic.academic.ru/dic.nsf/ruwiki/1029856), с помощью которого [программист](https://dic.academic.ru/dic.nsf/ruwiki/11604) мог задавать команды, оперируя с ячейками памяти, полностью используя возможности машины.

Машинный язык – это система команд конкретной [вычислительной машины](https://dic.academic.ru/dic.nsf/ruwiki/616), который [интерпретируется](https://dic.academic.ru/dic.nsf/ruwiki/15224) непосредственно [микропроцессором](https://dic.academic.ru/dic.nsf/ruwiki/1187537) или [микропрограммами](https://dic.academic.ru/dic.nsf/ruwiki/224644) данной вычислительной машины.

Однако использование большинства компьютеров на уровне машинного языка затруднительно. **(**Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких как номер блока на диске, номер сектора на дорожке и т. п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающие наличие и типы ошибок, которые надо анализировать.**)**

«Слова» на машинном языке называются инструкции, каждая из которых представляет собой одно элементарное действие для центрального процессора, такое, например, как считывание информации из ячейки памяти.

В случае, когда нужно иметь эффективную программу, вместо машинных языков используются близкие к ним машинно-ориентированные языки — [ассемблеры](https://dic.academic.ru/dic.nsf/ruwiki/5273). Команды языка ассемблера один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи команд и аргументов. Также язык ассемблера обеспечивает связывание частей программы и данныx через метки, выполняемое при ассемблировании (для каждой метки высчитывается адрес, после чего каждое вхождение метки заменяется на этот адрес). Но даже работа с ассемблером достаточно сложна и требует специальной подготовки.

**2. Подходы в разработке ПО. Спагетти-код, структурный подход, ООП, функциональщина.**

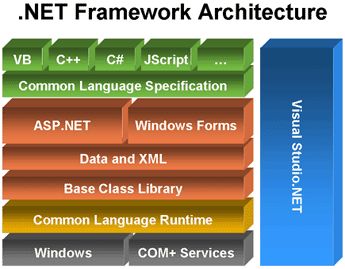
**3. Кроссплатформенность, переносимость, компиляция VS интерпретация и прочие причины возникновения Java.**

**4. C# и .NET Framework**

**.NET Framework** – платформа разработки, развертывания и выполнения распределенных приложений преимущественно в ОС Microsoft Windows. Она предоставляет объемный инструментарий для разработчиков, обеспечивает полную интеграцию языков, поддерживая межъязыковое наследование, обработку исключений и отладку. Программы, написанные на языках этой платформы запускаются не сами по себе, а в специальной среде (виртуальной машине) – **Common Language Runtime** – общеязыковая среда выполнения. Среда предоставляет такие функции, как управление памятью, отслеживание ошибок, проверка безопасности и другие.

В чем преимущество платформы .NET? Она позволяет разработчику уйти от большей части рутинной работы, автоматизируя ее, а именно - выделение памяти, ее освобождение (что было обязательно в C++). Также на ней можно разрабатывать и десктопные, и web-приложения, и приложения для мобильных устройств, и игры.

Далее на картинке можно увидеть приближенный взгляд на архитектуру платформы.

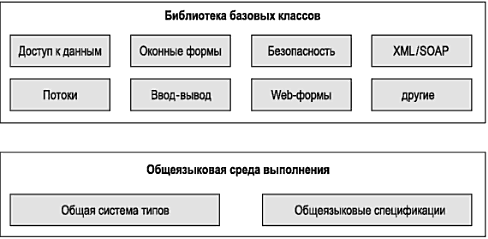


Все, что находится выше **Common Language Runtime**, контролируется средой, а та, в свою очередь, контролируется непосредственно операционной системой. Помимо среды выполнения, платформа предоставляет общие для всех своих языков технологии (выделены оранжевым), общеязыковую систему типов и общеязыковые спецификации.

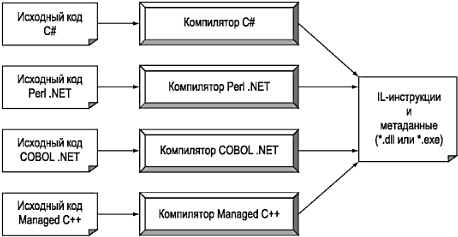
**Common Type System** – общеязыковая система типов. Спецификации CTS полностью описывают все возможные типы данных и программные конструкции, поддерживаемые средой выполнения, указывают, как эти элементы могут взаимодействовать друг с другом и как они представляются в формате метаданных .NET.

**Common Language Specification** – общеязыковые спецификации определяют подмножество общих типов и конструкций, понятных всем языкам, совместимым с платформой .NET. Типы данных, не выходящие за рамки CLS, будут поддерживаться всеми языками платформы.

Также платформа предоставляет библиотеку базовых классов, доступную всем языкам программирования .NET. Она не только инкапсулирует различные «примитивы», такие как потоки, файловый ввод-вывод, визуализация графики (и прочее), но и обеспечивает поддержку высокоуровневых сервисов.



Как же устроено межъязыковое взаимодействие? Код на любом языке .NET компилируется дважды: сперва компилятор конкретного языка переводит код в промежуточный язык – **Common Intermediate Language –** а затем IL-инструкции компилируются под конкретный процессор. Выполняющий эту задачу компилятор называют JIT-компилятором (just-in-time).



Важно понимать, что двойная компиляция влияет на производительность. Если вы разрабатываете ПО для старого железа с сильно ограниченным объемом памяти, платформа .NET будет уступать C++. Напротив, на компьютерах хотя бы средней производительности имеются достаточные для .NET Framework ресурсы.

Далее на картинке более детально показан механизм выполнения любой программы, написанной на/для платформы .NET.

Язык C#.

Язык C# был разработан корпорацией Microsoft в конце 90-х годов как часть общей стратегии .NET. Впервые он был выпущен в виде альфа-версии в середине 2000 года. Главным разработчиком C# был Андерс Хейльсберг — один из ведущих в мире специалистов по языкам программирования, в 80-е годы он был автором очень удачной и имевшей большое значение разработки — языка Turbo Pascal, изящная реализация которого послужила образцом для создания всех последующих компиляторов.

Итак, C# - высокоуровневый, компилируемый (дважды) язык программирования высокого уровня, спроектированный и разработанный для взаимодействия с платформой .NET и призванный устранить множество неудобств, с которыми сталкивались разработчики/программисты в 80-90-е годы.

