***Прикладное программирование***

К прикладным программам можно отнести графические редакторы, браузеры, плееры, текстовые редакторы и т.д. Требования к скорости таких программ не столь критичны, поэтому в этой области можно применить как старый добрый C/C++, так и что-то более «человечнее» - Java, Object Pascal, Python. О последнем мы поговорим чуть позже, сейчас только скажу, что Python хоть и является скриптовым языком программирования, написания оконных приложений на нем также практикуется.  
Плюсом Java является кроссплатформенность приложений. Написанные на этом языке программы транслируются в промежуточный код, который исполняется виртуальной машиной Java (JVM). Сегодня на Java можно написать как апплет для ПК, так и серверное приложение. Написание приложений для ОС Android также осуществляется на этом языке. Нужно также упомянуть, что профессия Java-программиста является одной из наиболее востребованных.  
Delphi (так иногда называют Object Pascal) отлично подходит для быстрого написания прикладных приложений. Компилируется в машинный код, что делает его немного быстрее, нежели Java. Название Delphi не совсем корректно, так как оно означает не язык а среду разработки под OS Windows. А так как на этом языке можно писать под UNIX-подобные ОС и консольные приставки (например в среде Kylix или Lazarus), то правильнее именовать этот язык Object Pascal-ем.

***Скриптовые языки***

В рамках этой статьи рассматривается 3 основных скриптовых языка: PHP, Python и Perl.  
Язык PHP применяется в основном для написания веб-приложений (для этого он и был создан). Очень много начинающих программистов выбирают именно этот язык первым для изучения. Отсутствие строгой типизации переменных, большая библиотека стандартных функцию + PEAR сделали PHP одним из самых популярных скриптовых языков. По поводу того стоит ли выбирать этот язык первым для изучении ответить сложно: с одной стороны его дружелюбие к новичкам, с другой — плохой тон написания приложений.  
Вторым в списке скриптовых языков является Python. Как упоминалось выше, область его применения достаточно широка: начиная веб-приложениями, заканчивая консольными и GUI. По сравнению с PHP, Python имеет на порядок большую скорость работы. Изучив Python, вы не будете ограничивать себя какой-то одной областью программирования.  
В отличии от своих братьев, Perl теряет свою актуальность ввиду сравнительно низкой скорости работы. Имея в своем арсенале большой механизм регулярных выражений, Perl незаменим при разборе текстовой информации. И хоть на нем также можно создавать приложения для веб, он практически вытеснен PHP с Python-ом.

* Поддержка нескольких языков. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому мы можем сделать отдельные модули одного приложения на отдельных языках.
* Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент .NET Framework поддерживается на большинстве современных ОС Windows (Windows 10/8.1/8/7/Vista). А благодаря проекту Mono можно создавать приложения, которые будут работать и на других ОС семейства Linux, в том числе на мобильных платформах Android и iOS.
* Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.
* Разнообразие технологий. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF. Для создания веб-сайтов - ASP.NET и т.д.
* **Скорость разработки**
* С# позволяет стартовать разработку быстрее, а это позволяет быстрее получить прототип решения. Скорость разработки на С# на начальных этапах проекта значительно выше по сравнению с С++.   
  Однако, когда инфраструктура проекта создана, основные подходы и библиотеки выбраны, а билд настроен, скорость разработки на С++ и скорость разработки на С# становятся примерно одинаковыми.   
  Таким образом, в коротких малобюджетных проектах С# будет иметь преимущество по скорости разработки, но в длинных и относительно дорогих данное преимущество будет незначительным.
* **2. Кросплатформенность**
* С++ кросплатформенный по факту, хотя и с некоторыми оговорками, дополнительными затратами, а также бинарной несовместимость между платформами.  
  C#, по факту, оказался не кросплатформенный, несмотря на существование неофициальных .net окружений под разными платформами и даже потенциальную бинарную совместимость между платформами.  
    
  C# спроектирован быть кросплатформенным, однако его развитие не пошло в этом направлении. Поэтому под Windows образовалась достаточно полная .net инфраструктура; на других же платформах равноценной инфраструктуры не появилось.  
    
  При этом для разработки на С++ сложилась практически равноценная инфраструктура на большинстве существующих платформ, есть масса библиотек, которые скомпилированы или могут быть скомпилированы под любые существующие платформы. Именно поэтому существует огромное количество кросплатформенных приложений и библиотек разного масштаба, написанных на С++, на ряду с кросплатформенными библиотеками есть и библиотеки специфичные для отдельных платформ. Все это дает практический равноценные шансы для развития приложений на различных платформах.  
    
  И хотя С# возможно использовать для построения приложений под не-Windows платформы, проблемы, вызываемые использованием .net в не-Windows окружении, сводят на нет многие преимущества выбора C#. Поэтому рекомендовать его для кроссплатформенного использования можно разве что если код на C# уже написан. При этом надо четко понимать, что в перспективе это будет приносить дополнительные затраты на поддержку.
* **3. Производительность кода и требовательность к ресурсам**
* Очевидным является факт того, что возможности по оптимизации *unmanaged* кода куда шире, чем возможности по оптимизации managed кода. Таким образом, пиковая производительность кода достижима только в *unmanaged* исполнении, т.е. в пределе, почти любая задача на С++ может быть решена с меньшими требованиями к ресурсам. Поэтому в тяжелых задачах, связанных с обработкой большого количества данных, С++ имеет сильные преимущества перед С#.  
    
  Но стоит понимать, что при выборе неправильного подхода, на С++ вполне можно написать код, который будет работать медленнее кода на C#, выполняющего туже задачу.   
    
  Если говорить о совокупности субъективных «простоты разработки», «красоты кода» и объективной производительности, то используя C# проще написать код, удовлетворяющий этим критериям одновременно. Однако это не значит, что производительный код на С++ обязательно будет страшным или сложным для восприятия, просто при его написании потребуется более «творческий» подход для удовлетворения перечисленных критериев одновременно.  
    
  Фундаментальные основы преимуществ С++ в возможности писать код, который будет выполняться непосредственно процессором, и возможности прямой работы с памятью. Конечно, свобода дает больше возможностей создать себе проблемы, но в ряде случаев это лучше, чем невозможность преодоления потолка производительности. И этот потолок вполне может привести, например, к тому, что под решение задачи, для которого бы хватило одного хорошего сервера, вам придется собирать ферму из нескольких серверов, или же к тому, что ваше приложение будет требовать «топового» железа на задачи, для которых хватило бы железа выпущенного лет 7-10 назад.
* **4. Библиотеки**
* Отличие ассортимента С++ и С# библиотек в том, что С++ библиотек больше, они имеют большую историю, за которую стали неплохо отлажены и оптимизированы, часто кросплатформенны, многие с открытым кодом. Однако при всех положительных сторонах С++ библиотеки как имеют очень разную, часто даже архаичную архитектуру, часто не объектный, а структурно-процедурный интерфейс. Связано это с тем, что многое С++ библиотеки это С библиотеки.  
    
  Другая неприятная особенность С++ библиотек — это создание и переопределение своих базовых типов. Многие С++ библиотеки заводят свои типы строк, контейнеров, переопределяют некоторые базовые типы. Этому есть логичные объяснения (лучшая производительность, поддержка кросплатформенности, отсутствие подходящих типов на момент написание библиотеки), однако все это не добавляет удобства использования и красоты коду. Базовые же С++ библиотеки дают не так много, как дают стандартные библиотеки С#, поэтому подбор правильных библиотек для проекта С++ — это задача, необходимая даже в сравнительно простых проектах.   
    
  Однако не все так страшно, для С++ есть немало и «красивых» библиотек. Да и базовые библиотеки постоянно расширяются. Вопрос часто лишь в том, каких жертв потребует данная красота.  
    
  В С# перечисленных выше проблем значительно меньше. Огромное количество библиотек с .net идет в базе, плюс к ним множество свободно доступных библиотек, это покрывает практически все первостепенные задачи разработки под Windows. Наличие большого количества стандартных типов почти избавляет от библиотек, где базовые типы переопределены. И в силу того, что библиотеки С# сравнительно молодые,- интерфейсы библиотек, как правило, лучше вписываются в те или иные шаблоны проектирования, что часто упрощает их изучение.  
    
  Однако же при ближайшем рассмотрении велик шанс, что под вашу специфическую задачу С# библиотеки не окажется, более того, может оказаться, что и решать такую задачу на С# достаточно не эффективно, поэтому подобной библиотеки не появится и в будущем, а если и появится, то будет работать недостаточно быстро.  
    
  Вторая неприятная особенность библиотек С# в том, что многие из них являются просто оберткой над *unmanaged* библиотеками, что будет всегда приводить к потерям производительности на конверсиях типов, и создавать дополнительные проблемы отладки и распространения.
* **6. Язык и Синтаксис**
* С первого взгляда код С++ и С# очень похож внешне. Но многообразие кода на С++ больше, ведь С++ является одновременно и С и С++ и С++0х и все это вы можете использовать одновременно (конечно, если это поддерживает ваш компилятор).  
    
  С# же, это только C#, хотя его синтаксис постоянно расширяется. Код на С#, как правило, выглядит проще и лаконичнее, чем код С++ (хотя это не всегда можно было сказать про первые версии С#). Языковые конструкции С++ и С# очень схожи, однако существенные различия можно найти в деталях.   
  Если С++ можно упрекнуть за отсутствие «в базе» reflection, позднего связывания и сборки мусора. То С# надо упрекнуть за отсутствие полноценных деструкторов, отсутствие полноценных макросов, достаточно грубую настройку наследования, отсутствие константных методов и членов, отсутствие глобальным методов (процедур), очень ограниченную поддержку шаблонов, список можно продолжать… Однако жить без всего этого вполне можно как в случае С++, так и в случае C#.  
    
  Синтаксис С#, пожалуй, можно назвать упрощенной версией С++, таким образом С#, как и любое упрощение, одновременно несет и позитивный и негативный эффекты.  
    
  Стоит сказать, что более сложный код часто легче пишется и анализируется, если написан более простым языком. С этой позиции, используя С#, меньше шансов допустить ошибку в принципиально сложном коде и больше шансов написать чистый код, обладая теми же ресурсами. Это может быть полезно при решении достаточно сложных, но не требовательных к производительности задач. Однако при этом большее количество «синтетики» в С# делает меньше оценку производительности кода по его «внешнему виду».