Введение
История создания систем и языков программирования

По мере развития вычислительной техники возникали различные методики программирования. На каждом этапе создавался новый подход, который позволял программистам справляться с растущим усложнением программ.

1883 год: алгоритм для аналитической машины – Создан Адой Лавлейс для аналитической машины Чарльза Беббиджа для вычисления чисел Бернулли, он считается первым языком программировани.

Первые программы создавались по средствам ключевых переключателей на передней панели компьютера. Очевидно, что такой способ подходит для очень небольших программ.

В 50-х годах (**1949**) появились первые средства автоматизации программирования – это языки **автокоды (1952) и ассемблеры**.

Автокод – был общим термином для семейства ранних языков программирования. Первый был разработан Аликом Гленни для компьютера Mark1 в Манчестерском университете в Великобритании. Некоторые считают автокод первым компилируемым языком программирования.

С появлением этих языков переменные величины стали обозначаться символами, числовые коды операций заменились на мнемонические обозначения. При этом язык программирования стал понятнее программисту, но удалился от языка машинных команд. Чтобы компьютер мог использовать программу, написанную на таком языке, потребовался специальный переводчик – транслятор.

Транслятор – системная программа, переводящая текст программы в машинные коды.

Заложенный в транслятор алгоритм перевода сложен. При этом существует два основных способа трансляции – **компиляция** программы или ее **интерпретация**.

При компиляции весь исходный программный код с разу переводится в машинный. Создается отдельный **исполняемый файл**, который никак не связан с исходным кодом. Выполнение исполняемого файла обеспечивается операционной системой. После того, как получен исполняемый файл, для его чтения транслятор уже не нужен.

При **интерпретации** выполнение кода происходит последовательно (условно можно сказать, строка за строкой). Грубо говоря, операционная система взаимодействует с интерпретатором, а не с файлом, содержащим программный код. Интерпретатор, прочитав очередную часть исходного кода, переводит его в машинный код и отдает его операционной системе. Операционная система исполняет этот код и ждет следующего фрагмента от интерпретатора (характерно для языка Питон).

Выполнение откомпилированной программы происходит быстрее, т.к. она представляет собой готовый машинный код. Однако на современных компьютерах снижение скорости выполнения при интерпретации обычно не заметно.

Языки типа ассемблер являются машинно-ориентированными, т.е. они настроены на структуру команд конкретного компьютера. Разные компьютеры с разными процессорами имеют различные ассемблеры.

Ассемблеры на сегодняшний день продолжают использоваться. В системном программировании с их помощью создаются низкоуровневые интерфейсы операционных систем, компоненты драйверов.

После ассемблеров наступил расцвет языков так называемого **высокого уровня**. Для них потребовалось разрабатывать более сложные трансляторы, т.к. языки высокого уровня более удобны для человека, чем для машины.

В отличие от ассемблеров, которые остаются привязанными к своим типам машин, языки высокого уровня обладают **переносимостью**.

Первыми популярными языками высокого уровня, появившимися в 50-х годах, были – **Фортран (1957), Кобол (1959), Алгол (1958).** Программисты могли писать программы до нескольких тысяч строк длиной. Фортран и Алгол были ориентированы на научно-технические расчеты математического характера. Алгол послужил отправной точкой в разработке языков Паскаль, C, C++ и Ява.

Кобол применялся для решения экономических задач (Он используется в банкоматах, обработке кредитных карт, телефонных системах и т.д.). Языки высокого уровня являются машинно-независимыми. И форма записи на таком языке ближе к математической.

Для того времени указанный подход к программированию был наиболее перспективным. Однако, языки программирования легко понимались в коротких программах, когда дело касалось больших программ – становились нечитабельными.

Избавление от таких неструктурированных программ пришло после изобретения языков структурного программирования – 60-е годы (Паскаль, С).

В 1964 году был разработан **Basic**. В эпоху ЭВМ 3-го поколения большое распространение получил язык PL/1. Это был первый язык претендующий на универсальность, т.е. на возможность решать любые задачи. Однако, он оказался слишком сложным и практически не использовался.

В 1971 году швейцарским профессором Николасом Виртом был разработан язык **Pascal**, как учебный язык структурного программирования. Позднее фирма Borland разработала систему программирования Turbo Pascal – это не только язык и транслятор с него, но и операционная оболочка. В силу своих достоинств Pascal вышел за рамки учебного языка и стал языком профессионального программирования, на основе которого были разработаны такие языки, как **Ода (модифицированный Basic), Модула-2, Delphi**.

Язык программирования **С** создавался как инструментальный язык для разработки операционных систем, трансляторов, баз данных. В отличие от Pascal в нем заложены возможности непосредственного обращения к некоторым машинным командам, а также определенным участкам памяти компьютера. В развитии С пришел к объектно-ориентированному языку **С++. С++** был разработан сотрудником научно-исследовательского центра AT&T Bell Laboratories Бьярном Страуструпом в 1979г. Первоначально название «С с классами». В 1983г. Было изменено на С++.

ЭВМ 5-го поколения называют машинами искусственного интеллекта. Языками программирования для таких машин являются **Лисп и Пролог**. Лисп появился в 1965 году. Он основан на понятии рекурсивного определения функции. С его помощью можно моделировать достаточно сложные процессы, в частности интеллектуальную деятельность человека.

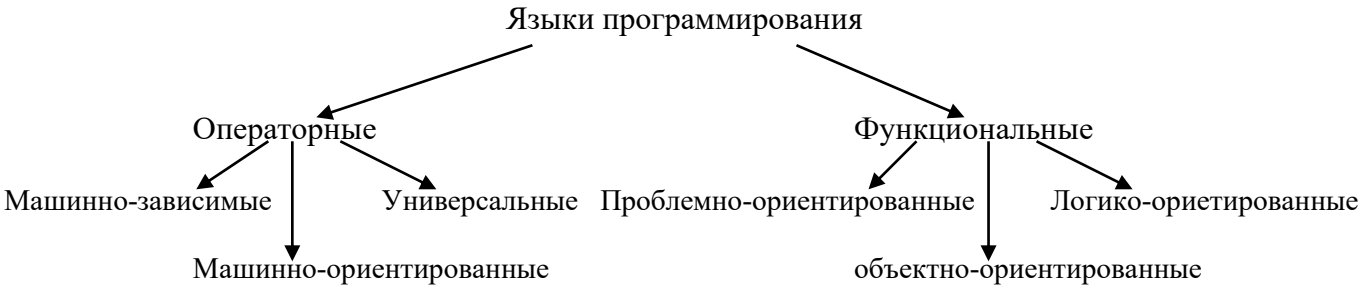
Пролог был разработан во Франции в 1972 году. Используется для решения проблем искусственного интеллекта. Пролог позволяет в формальном виде описывать различны утверждения, логику рассуждений, а также получать от ЭВМ ответы на заданные вопросы.

1991: Python, Visual Basic, 1995: Java, PHP, Ruby, JavaScript, 2000: C#, 2014: Swift.

Структурное программирование подразумевало точно обозначенные управляющие структуры, программные блоки, отсутствие или минимальное использование инструкции GOTO, автономные подпрограммы, в которых поддерживалась рекурсия и локальные переменные.

Сутью структурного программирования являлась возможность разбиения программы на составляющие ее элементы. Используя структурное программирование, средний программист может создавать и поддерживать программы свыше 50 тысяч строк длиной.

Хотя структурное программирование, при его использовании для умеренно сложных программ, принесло выдающиеся результаты, даже оно оказалось несостоятельным тогда, когда программа достигала определенной длины. В итоге были разработаны принципы ООП.



К машинно-зависимым относятся – ассемблеры; к машинно-ориентированным – С и С++; универсальные – Бейсик, Паскаль, Фортран, Кобол; логико-ориентированные – Пролог, Лисп.

ООП аккумулирует лучшие идеи воплощенные в структурном программировании и сочетает их с новыми мощными концепциями, которые позволяют оптимально организовывать программы.

ООП позволяет разложить проблему на составные части, причем каждая составная становится самостоятельным объектом, содержащим свои коды и данные, которые относятся к этому объекту.

В этом случае вся процедура в целом упрощена и программист получает возможность оперировать с гораздо большими по объему программами.

Все языки ООП, включая C++, основаны на трех основополагающих концепциях:

- инкапсуляция;
- полиморфизм;
- наследование.

Инкапсуляция – это механизм, который объединяет данные и код, манипулирующий с этими данными, а так же защиту от того и другого от внешнего вмешательства и неправильного использования. Внутри объекта коды и данные могут быть закрытыми для этого объекта или открытыми.

Полиморфизм – свойство, которое позволяет одно и тоже имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма является использование одного имени для задания общих для класса действий. Выполнение каждого действия будет определено типом данных. Например, для языка С, в котором полиморфизм поддерживается недостаточно, нахождение абсолютной величины числа требует трех различных функций (для целых, длинных целых, чисел с плавающей точкой). В C++ можно использовать одно имя функции для множества различных действий – это называется перегрузкой функции.

Полиморфизм может применятся также и к операторам и называется перегрузкой оператора. В более общем смысле концепцией полиморфизма является идея: один интерфейс – множество методов. Это означает, что можно создать один интерфейс для группы близких по смыслу действий. Выбор же конкретного действия в зависимости от ситуации возлагается на компилятор.

Наследование – процесс, по средствам которого один объект может приобретать свойства другого объекта и добавлять к ним черты характерные только для него.

Наследование является важным, поскольку оно позволяет поддержать концепцию иерархии классов, применение которых делает управляемыми большие потоки информации. Без использования иерархии классов для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающи его определяли. Однако при использовании наследования можно описать объект путем определения того общего класса, к которому он относится, с теми специфическими чертами, которые делают объект уникальным.

Язык программирования C++

Язык программирования C++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений.

Своими корнями он уходит в язык Си, который был разработан в 1969—1973 годах в компании Bell Labs программистом Деннисом Ритчи (Dennis Ritchie). В начале 1980-х годов датский программист Бьерн Страуструп (Bjarne Stroustrup), который в то время работал в компании Bell Labs, разработал C++ как расширение к языку Си. Фактически вначале C++ просто дополнял язык Си некоторыми возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как "C with classes" ("Си с классами").

Впоследствии новый язык стал набирать популярность. В него были добавлены новые возможности, которые делали его не просто дополнением к Си, а совершенно новым языком программирования. В итоге "Си с классами" был переименован в C++. И с тех по оба языка стали развиваться независимо друг от друга.
C++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко C++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на C++. Но только системным программированием применение данного языка не ограничивается. C++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Нередко он применяется для создания графических приложений, различных прикладных программ. Также особенно часто его используют для создания игр с богатой насыщенной визуализацией. Кроме того, в последнее время набирает ход мобильное направление, где C++ тоже нашел свое применение. И даже в веб-разработке также можно использовать C++ для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем C++ - язык широкого пользования, на котором можно создавать практически любые виды программ.

C++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на C++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на C++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на C++ в приложения под эти платформы.

В отличие от Си язык C++ позволяет писать приложения в объектно-ориентированном стиле, представляя программу как совокупность взаимодействующих между собой классов и объектов. Что упрощает создание крупных приложений.

Основные этапы развития

В 1979-80 годах Бьерн Страуструп разработал расширение к языку Си - "Си с классами". В 1983 язык был переименован в C++.

В 1985 году была выпущена первая коммерческая версия языка C++, а также первое издание книги "Языка программирования C++", которая представляла первое описание этого языка при отсутствии официального стандарта.

В 1989 была выпущена новая версия языка C++ 2.0, которая включала ряд новых возможностей. После этого язык развивался относительно медленно вплоть до 2011 года. Но при этом в 1998 году была предпринята первая попытка по стандартизации языка организацией ISO (International Organiztion for Standartization). Первый стандарт получил название ISO/IEC 14882:1998 или сокращенно C++98. В дальнейшем в 2003 была издана новая версия стандарта C++03.

В 2011 году был издан новый стандарт C++11, который содержал множество добавлений и обогащал язык C++ большим числом новых функциональных возможностей. После этого в 2014 году было выпущено небольшое добавление к стандарту, известное также как C++14. И еще один ключевой релиз языка намечен на 2017.

Компиляторы и среды разработки

Для разработки программ на C++ необходим компилятор - он транслирует исходный код на языке C++ в исполняемый файл, который затем можно запускать. Но в настоящий момент есть очень много различных компиляторов. Они могут отличаться по различным аспектам, в частности, по реализации стандартов. Рекомендуется для разработки выбирать те компиляторы, которые развиваются и реализуют все последние стандарты. Так, на протяжении всего руководства преимущественно будет использоваться свободно распространяемый компилятор **g++**, разработанный в рамках проекта GNU.

Также для создания программ можно использовать интегрированные среды разработки IDE, такие как Visual Studio, Netbeans, Eclipse, Qt и т.д.