Языки программирования. Классификация (императивные, декларативные, параллельные) и примеры. Компиляторы и интерпретаторы. Объектно-ориентированное программирование

Для подготовки по данному вопросу можно рекомендовать следующую работу:

- Т. Пратт, М. Зелковиц. Языки программирования: разработка и реализация. 4-е изд. СПб.:Питер, 2003

Краткая информация приведена ниже.

Язык программирования — формальная знаковая система, предназначенная для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, задающих внешний вид программы и действия, которые выполнит исполнитель (компьютер) под ее управлением.

Со времени создания первых программируемых машин человечество придумало более двух с половиной тысяч языков программирования. Каждый год их число пополняется новыми. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

Стандартизация языков программирования

Язык программирования может быть представлен в виде набора спецификаций, определяющих его синтаксис и семантику.

Для многих широко распространённых языков программирования созданы международные стандарты. Специальные организации проводят регулярное обновление и публикацию спецификаций и формальных определений соответствующего языка. В рамках таких комитетов продолжается разработка и модернизация языков программирования и решаются вопросы о расширении или поддержке уже существующих и новых языковых конструкций.

Типы данных

Современные цифровые компьютеры обычно являются двоичными и данные хранят в двоичном (бинарном) коде (хотя возможны реализации и в других системах счисления). Эти данные как правило отражают информацию из реального мира (имена, банковские счета, измерения и др.), представляющую высокоуровневые концепции.

Особая система, по которой данные организуются в программе, — это система типов языка программирования; разработка и изучение систем типов известна под названием теория типов. Языки могут быть классифицированы как системы со статической типизацией и языки с динамической типизацией.

Статически-типизированные языки могут быть в дальнейшем подразделены на языки с обязательной декларацией, где каждая переменная и объявление функции имеет обязательное объявление типа, и языки с выводимыми типами. Иногда динамически-типизированные языки называются латентно-типизированными.

Структуры данных

Системы типов в языках высокого уровня позволяют определять сложные, составные типы, так называемые структуры данных. Как правило, структурные типы данных образуются как декартово произведение базовых (атомарных) типов и ранее определённых составных типов.

Основные структуры данных (списки, очереди, хеш-таблицы, двоичные деревья и пары) часто представлены особыми синтаксическими конструкциями в языках высокого уровня. Такие данные структурируются автоматически.

Парадигма программирования

Язык программирования строится в соответствии с той или иной базовой моделью вычислений и парадигмой программирования.

Несмотря на то, что большинство языков ориентировано на императивную модель вычислений, задаваемую фон-неймановской архитектурой ЭВМ, существуют и другие подходы. Можно упомянуть языки со стековой вычислительной моделью (Forth, Factor, Postscript и др.), а также функциональное (Лисп, Haskell, ML и др.) и логическое программирование (Пролог) и язык Рефал, основанный на модели вычислений, введённой советским математиком А. А. Марковым-младшим.

В настоящее время также активно развиваются проблемно-ориентированные, декларативные и визуальные языки программирования.

Способы реализации языков

Языки программирования могут быть реализованы как *компилируемые* и *интерпретируемые*.

Программа на компилируемом языке при помощи специальной программы компилятора преобразуется (компилируется) в набор инструкций для данного типа процессора (машинный код) и далее записывается в исполнимый модуль, который может быть запущен на выполнение как отдельная программа. Другими словами, компилятор переводит исходный текст программы с языка программирования высокого уровня в двоичные коды инструкций процессора.

Если программа написана на интерпретируемом языке, то интерпретатор непосредственно выполняет (интерпретирует) исходный текст без предварительного перевода. При этом программа остаётся на исходном языке и не может быть запущена без интерпретатора. Можно сказать, что процессор компьютера — это интерпретатор машинного кода.

Кратко говоря, компилятор переводит исходный текст программы на машинный язык сразу и целиком, создавая при этом отдельную машинно-исполняемую программу, а интерпретатор выполняет исходный текст прямо во время исполнения программы («интерпретируя» его своими средствами).

Разделение на компилируемые и интерпретируемые языки является условным. Так, для любого традиционно компилируемого языка, как, например, Паскаль, можно написать интерпретатор. Кроме того, большинство современных «чистых» интерпретаторов не исполняют конструкции языка непосредственно, а компилируют их в некоторое высокоуровневое промежуточное представление (например, с разыменованием переменных и раскрытием макросов).

Для любого интерпретируемого языка можно создать компилятор — например, язык Лисп, изначально интерпретируемый, может компилироваться без каких бы то ни было ограничений. Создаваемый во время исполнения программы код может так же динамически компилироваться во время исполнения.

Как правило, скомпилированные программы выполняются быстрее и не требуют для выполнения дополнительных программ, так как уже переведены на машинный язык. Вместе с тем, при каждом изменении текста программы требуется её перекомпиляция, что создаёт трудности при разработке. Кроме того, скомпилированная программа может выполняться только на том же типе компьютеров и, как правило, под той же операционной системой, на которую был рассчитан компилятор. Чтобы создать исполняемый файл для машины другого типа, требуется новая компиляция.

Интерпретируемые языки обладают некоторыми специфическими дополнительными возможностями (см. выше), кроме того, программы на них можно запускать сразу же после изменения, что облегчает разработку. Программа на интерпретируемом языке может быть зачастую запущена на разных типах машин и операционных систем без дополнительных усилий.

Однако интерпретируемые программы выполняются заметно медленнее, чем компилируемые, кроме того, они не могут выполняться без дополнительной программы-интерпретатора.

Некоторые языки, например, Java и C#, находятся между компилируемыми и интерпретируемыми. А именно, программа компилируется не в машинный язык, а в машинно-независимый код низкого уровня, байт-код. Далее байт-код выполняется виртуальной машиной. Для выполнения байт-кода обычно используется интерпретация, хотя отдельные его части для ускорения работы программы могут быть транслированы в машинный код непосредственно во время выполнения программы по технологии компиляции «на лету» (Just-in-time compilation, JIT). Для Java байт-код исполняется виртуальной машиной Java (Java Virtual Machine, JVM), для C# — Common Language Runtime.

Подобный подход в некотором смысле позволяет использовать плюсы как интерпретаторов, так и компиляторов. Следует упомянуть также язык Форт(Forth), имеющий и интерпретатор, и компилятор.

Классификация языков и подходов к программированию.

Первые языки программирования возникли относительно недавно. Различные исследователи указывают в качестве времени их создания 20-е, 30-е и даже 40-е годы XX столетия. Нашей задачей является не установление самого раннего языка, а поиск закономерностей в их развитии.

Как и следовало ожидать, первые языки программирования, как и первые ЭВМ, были довольно примитивны и ориентированы на численные расчеты. Это были и чисто теоретические научные расчеты (прежде всего, математические и физические), и прикладные задачи, в частности, в области военного дела.

Программы, написанные на ранних языках программирования, представляли собой линейные последовательности элементарных операций с регистрами, в которых хранились данные.

Нужно отметить, что ранние языки программирования были оптимизированы под аппаратную архитектуру конкретного компьютера, для которого предназначались, и хотя они обеспечивали

высокую эффективность вычислений, до стандартизации было еще далеко. Программа, которая была вполне работоспособной на одной вычислительной машине, зачастую не могла выполняться на другой.

Таким образом, ранние языки программирования существенно зависели от того, что принято называть средой вычислений и приблизительно соответствовали современным машинным кодам или языкам ассемблера.

Следующее десятилетие ознаменовалось появлением языков программирования так называемого "высокого уровня", по сравнению с ранее рассмотренными предшественниками, соответственно именуемыми низкоуровневыми языками.

При этом различие состоит в повышении эффективности труда разработчиков за счет абстрагирования от конкретных деталей аппаратного обеспечения. Одна инструкция (оператор) языка высокого уровня соответствовала последовательности из нескольких низкоуровневых инструкций, или команд. Исходя из того, что программа, по сути, представляла собой набор директив, обращенных к компьютеру, такой подход к программированию получил название императивного.

Еще одной особенностью языков высокого уровня была возможность повторного использования ранее написанных программных блоков, выполняющих те или иные действия, посредством их идентификации и последующего обращения к ним, например по имени. Такие блоки получили название функций или процедур, и программирование приобрело более упорядоченный характер.

Кроме того, с появлением языков высокого уровня зависимость реализации от аппаратного обеспечения существенно уменьшилась. Платой за это стало появление специализированных программ, преобразующих инструкции языков в коды той или иной машины, или трансляторов, а также некоторая потеря в скорости вычислений, которая, впрочем, компенсировалась существенным выигрышем в скорости разработки приложений и унификацией программного кода.

Нужно отметить, что операторы и ключевые слова новых языков программирования были более осмысленными, чем безликие цифровые последовательности кодов, что также обеспечивало повышение производительности труда программистов.

Естественно, для обучения новым языкам программирования требовалось много времени и средств, а эффективность реализации на прежнем аппаратном обеспечении снижалась. Однако это были временные трудности, и, как показала практика программирования, многие из первых языков высокого уровня оказались настолько удачно реализованными, что активно используются и сегодня.

Одним из таких примеров является язык Fortran, реализующий вычислительные алгоритмы. Другой пример – язык APL, трансформировавшийся в BPL и затем в C. Основные конструкции последнего остаются неизменными вот уже несколько десятилетий и присутствуют в таких языках, как Java и C#.

Примеры других языков программирования: ALGOL, COBOL, Pascal, Basic.

В 60-х годах возникает новый подход к программированию, который до сих пор успешно конкурирует с императивным, а именно, декларативный подход.

Суть подхода состоит в том, что программа представляет собой не набор команд, а описание действий, которые необходимо осуществить.

Этот подход, как мы увидим впоследствии, существенно проще и прозрачнее формализуется математическими средствами. Следовательно, программы проще проверять на наличие ошибок (тестировать), а также на соответствие заданной технической спецификации (верифицировать).

Высокая степень абстракции также является преимуществом данного подхода. Фактически, программист оперирует не набором инструкций, а абстрактными понятиями, которые могут быть достаточно обобщенными.

На начальном этапе развития декларативным языкам программирования было сложно конкурировать с императивными в силу объективных трудностей эффективной реализации трансляторов. Программы работали медленнее, однако они могли решать более абстрактные задачи с меньшими трудозатратами.

В частности, язык SML был разработан как средство доказательства теорем.

Различные диалекты языка LISP (в частности, Interlisp, Common Lisp, Scheme), возникли потому, что ядро и идеология этого языка оказались весьма эффективными при реализации символьной обработки (анализе текстов).

Другие характерные примеры декларативных языков программирования: SML, Haskell, Prolog.

Одним из путей развития декларативного стиля программирования стал функциональный подход, возникший после создания языка LISP.

Отличительной особенностью данного подхода является то, что любая программа, написанная на таком языке, может интерпретироваться как функция с одним или несколькими аргументами. Такой подход дает возможность прозрачного моделирования текста программ математическими средствами, а значит, весьма интересен с теоретической точки зрения.

Сложные программы при таком подходе строятся посредством агрегирования функций. При этом текст программы представляет собой функцию, некоторые аргументы которой можно также рассматривать как функции. Таким образом, повторное использование кода сводится к вызову ранее описанной функции, структура которой, в отличие от процедуры императивного языка, математически прозрачна.

Более того, типы отдельных функций, используемых в функциональных языках, могут быть переменными. Таким образом обеспечивается возможность обработки разнородных данных (например, упорядочение элементов списка по возрастанию для целых чисел, отдельных символов и строк) или полиморфизм.

Еще одним важным преимуществом реализации языков функционального программирования является автоматизированное динамическое распределение памяти компьютера для хранения данных. При этом программист избавляется от обязанности контролировать данные, а при

необходимости может запустить функцию "сборки мусора" – очистки памяти от тех данных, которые больше не потребуются программе (обычно этот процесс периодически инициируется компьютером).

Таким образом, при создании программ на функциональных языках программист сосредотачивается на области исследований (предметной области) и в меньшей степени заботится о рутинных операциях (обеспечении правильного с точки зрения компьютера представления данных, "сборке мусора" и т.д.).

Поскольку функция является естественным формализмом для языков функционального программирования, реализация различных аспектов программирования, связанных с функциями, существенно упрощается. В частности, становится прозрачным написание рекурсивных функций, т.е. функций, вызывающих самих себя в качестве аргумента. Кроме того, естественной становится и реализация обработки рекурсивных структур данных (например, списков – базовых элементов, скажем, для языков семейства LISP, деревьев и др.)

Благодаря реализации механизма сопоставления с образцом, такие языки как ML и Haskell вполне применимы для символьной обработки.

Естественно, языки функционального программирования не лишены недостатков. Часто к ним относят нелинейную структуру программы и относительно невысокую эффективность реализации. Однако первый недостаток достаточно субъективен, а второй успешно преодолен современными реализациями, в частности, рядом последних трансляторов языка SML, включая и компилятор для среды Microsoft .NET.

В 70-х годах возникла еще одна ветвь языков декларативного программирования, связанная с проектами в области искусственного интеллекта, а именно языки логического программирования.

Согласно логическому подходу к программированию, программа представляет собой совокупность правил или логических высказываний. Кроме того, в программе допустимы логические причинно-следственные связи, в частности, на основе операции импликации.

Таким образом, языки логического программирования базируются на классической логике и применимы для систем логического вывода, в частности, для так называемых экспертных систем. На языках логического программирования естественно формализуется логика поведения, и они применимы для описаний правил принятия решений, например, в системах, ориентированных на поддержку бизнеса.

Важным преимуществом такого подхода является достаточно высокий уровень машинной независимости, а также возможность откатов – возвращения к предыдущей подцели при отрицательном результате анализа одного из вариантов в процессе поиска решения (скажем, очередного хода при игре в шахматы), что избавляет от необходимости поиска решения путем полного перебора вариантов и увеличивает эффективность реализации.

Одним из недостатков логического подхода в концептуальном плане является специфичность класса решаемых задач.

Другой недостаток практического характера состоит в сложности эффективной реализации для принятия решений в реальном времени, скажем, для систем жизнеобеспечения.

Нелинейность структуры программы является особенностью декларативного подхода и, строго говоря, представляет собой оригинальную особенность, а не объективный недостаток.

В качестве примеров языков логического программирования можно привести Prolog (название возникло от слов PROgramming in LOGic) и Mercury.

Важным шагом на пути к совершенствованию языков программирования стало появление объектно-ориентированного подхода к программированию (ООП) и соответствующего класса языков.

В рамках данного подхода программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов).

Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.

Еще одним теоретически интересным и практически важным свойством объектно-ориентированного подхода является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области.

Перемещаясь по иерархии классов от более общих понятий предметной области к более конкретным (или от более сложных – к более простым) и наоборот, программист получает возможность изменять степень абстрактности или конкретности взгляда на моделируемый им реальный мир.

Использование ранее разработанных (возможно, другими коллективами программистов) библиотек объектов и методов позволяет значительно сэкономить трудозатраты при производстве программного обеспечения, в особенности типичного.

Объекты, классы и методы могут быть полиморфными, что делает реализованное программное обеспечение более гибким и универсальным.

Сложность адекватной (непротиворечивой и полной) формализации объектной теории порождает трудности тестирования и верификации созданного программного обеспечения. Вероятно, это обстоятельство является одним из самых существенных недостатков объектно-ориентированного подхода к программированию.

Пожалуй, наиболее известным примером объектно-ориентированного языка программирования является язык C++, развившийся из императивного языка C. Наиболее востребоваными объектно-ориентированными языками программирования, помимо C++, являются на сегодняшний момент Java и C#.

Другие примеры объектно-ориентированных языков программирования: Visual Basic, Eiffel, Oberon.

Развитием событийно управляемой концепции объектно-ориентированного подхода стало появление в 90-х годах целого класса языков программирования, которые получили название языков сценариев или скриптов.

В рамках данного подхода программа представляет собой совокупность возможных сценариев обработки данных, выбор которых инициируется наступлением того или иного события (щелчок по кнопке мыши, попадание курсора в определенную позицию, изменение атрибутов того или иного объекта, переполнение буфера памяти и т.д.). События могут инициироваться как операционной системой, так и пользователем.

Основные достоинства языков данного класса унаследованы от объектно-ориентированных языков программирования. Это интуитивная ясность описаний, близость к предметной области, высокая степень абстракции, хорошая переносимость.

Широкие возможности повторного использования кода также унаследованы сценарными языками от объектно-ориентированных предков.

Существенным преимуществом языков сценариев является их совместимость с передовыми инструментальными средствами автоматизированного проектирования и быстрой реализации программного обеспечения, или так называемыми CASE- (Computer-Aided Software Engineering) и RAD- (Rapid Application Development) средствами.

Естественно, что вместе с достоинствами объектно-ориентированного подхода языки сценариев унаследовали и ряд недостатков. К последним, прежде всего, относятся сложность тестирования и верификации программ и возможности возникновения в ходе эксплуатации множественных побочных эффектов, проявляющихся за счет сложной природы взаимодействия объектов и среды, представленной интерфейсами с большим количеством одновременно работающих пользователей программного обеспечения, операционной системой и внешними источниками данных.

Характерные примеры сценарных языков программирования: VBScript, PowerScript, LotusScript, JavaScript.

Еще один весьма важный класс языков программирования – языки поддержки параллельных вычислений.

Программы, написанные на этих языках, представляют собой совокупность описаний процессов, которые могут выполняться как в действительности одновременно, так и в псевдопараллельном режиме. В последнем случае устройство, обрабатывающее процессы, функционирует в режиме разделения времени, выделяя время на обработку данных, поступающих от процессов, по мере необходимости (а также с учетом последовательности или приоритетности выполнения операций).

Языки параллельных вычислений позволяют достичь заметного выигрыша при обработке больших массивов информации, поступающих от одновременно работающих пользователей, либо имеющих высокую интенсивность (как, например, видеоинформация или звуковые данные высокого качества).

Другой весьма значимой областью применения языков параллельных вычислений являются системы реального времени, в которых пользователю необходимо получить ответ от системы непосредственно после запроса. Такого рода системы отвечают за жизнеобеспечение и принятие ответственных решений.

Обратной стороной достоинств рассматриваемого класса языков программирования является высокая стоимость разработки программного обеспечения, следовательно, создание относительно небольших программ широкого (например, бытового) применения зачастую нерентабельно.

Примерами языков программирования с поддержкой параллельных вычислений могут служить Ada, Modula-2 и Oz.

Итак, в данной лекции были рассмотрены история и эволюция языков программирования и основные подходы к разработке программных систем. Была сделана попытка классификации языков и подходов к программированию, а также проведен анализ достоинств и недостатков, присущих тем или иным подходам и языкам.

Заметим, что приведенную классификацию не следует считать единственно верной, поскольку языки программирования постоянно развиваются и совершенствуются, и недавние недостатки устраняются с появлением необходимых инструментальных средств и теоретических обоснований.