30. Эволюция языков программирования

Автор: Александр
26.08.2014 14:51

Развитие вычислительной техники сопровождается созданием новых и совершенствованием существующих средств общения программистов с ЭВМ - языков программирования (ЯП).

Под ЯП понимают правила представления данных и записи алгоритмов их обработки, которые автоматически выполняются ЭВМ. В более абстрактном виде ЯП является средством создания программных моделей объектов и явлений внешнего мира.

К настоящему времени созданы десятки различных ЯП от самых примитивных до близких к естественному языку человека. Чтобы разобраться во всем многообразии ЯП, нужно знать их классификацию, а также историю создания, эволюцию и тенденции развития.

Чтобы понимать тенденции развития ЯП, нужно знать движущие силы их эволюции. Для выяснения этого вопроса будем рассматривать ЯП с различных точек зрения.

Во-первых, ЯП является инструментом программиста для создания программ. Для создания хороших программ нужны хорошие ЯП. Поэтому одной из движущих сил эволюции ЯП является стремление разработчиков к созданию более совершенных программ.

Во-вторых, процесс разработки программы можно сравнивать с промышленным производством, в котором определяющими факторами являются производительность труда коллектива программистов, себестоимость и качество программной продукции. Создаются различные технологии разработки программ (структурное, модульное, объектно-ориентированное программирование и другие), которые должны поддерживаться ЯП. Поэтому второй движущей силой эволюции ЯП является стремление к повышению эффективности процесса производства программной продукции.

В-третьих, программы можно рассматривать как аналог радиоэлектронных устройств обработки информации, в которых вместо радиодеталей и микросхем используют конструкции ЯП (элементная база программы). Как и электронные устройства, программы могут быть простейшими (уровня детекторного приемника) и очень сложными (уровня автоматической космической станции), при этом уровень инструмента должен

соответствовать сложности изделия. Кроме того, человеку удобнее описывать моделируемый объект в терминах предметной области, а не языком цифр. Поэтому третьей движущей силой, ведущей к созданию новых, специализированных, ориентированных на проблемную область и более мощных ЯП, является увеличение разнообразия и повышение сложности задач, решаемых с помощью ЭВМ.

В-четвертых, совершенствование самих ЭВМ приводит к необходимости создания языков, максимально реализующих новые возможности ЭВМ.

В-пятых, программы являются интеллектуальным продуктом, который нужно накапливать и приумножать. Но программы, как и технические изделия, обладают свойством морального старения, одной из причин которого является их зависимость от типа ЭВМ и операционной среды. С моральным старением программ борются путем их модернизации и выпуска новых версий, однако при высокой динамике смены типов ЭВМ и операционных сред разработчики будут только тем и заниматься, что модернизировать старые программы. Поэтому, ЯП должен обеспечивать продолжительный жизненный цикл программы, и стремление к этому является пятой движущей силой развития ЯП.

Известно, что первым программистом была женщина - леди Ада Лавлейс, дочь лорда Байрона. Она разрабатывала программы для одного из первых механических компьютеров, созданного в начале XIX века английским ученым Чарльзом Беббиджом. Однако настоящее программирование в современном понимании началось с момента создания первой электронной вычислительной машины. Но теме не менее, имя этой замечательной женщины - Ada - присвоено одному из самых мощных современных ЯП, который является базовым для министерства обороны США.

Первые ЭВМ, созданные человеком, имели небольшой набор команд и встроенных типов данных, но позволяли выполнять программы на машинном языке. Машинный язык (МЯ) - единственный язык, понятный ЭВМ. Он реализуется аппаратно: каждую команду выполняет некоторое электронное устройство. Программа на МЯ представляет собой последовательность команд и данных, заданных в цифровом виде. Например, команда вида 1A12 в 16-ричном виде или 0001101000010010 в двоичном виде означает операцию сложения (1A) содержимого регистров 1 и 2.

Данные на МЯ представлены числами и символами. Операции являются элементарными

и из них строится вся программа. Ввод программы в цифровом виде производился непосредственно в память с пульта ЭВМ либо с примитивных устройств ввода. Естественно, что процесс программирования был очень трудоемким, разобраться в программе даже автору было довольно сложно, а эффект от применения ЭВМ был довольно низким. Этот этап в развитии ЯП показал, что программирование является сложной проблемой, трудно поддающейся автоматизации, но именно программное обеспечение определяет в конечном счете эффективность применения ЭВМ. Поэтому на всех последующих этапах усилия направлялись на совершенствование интерфейса между программистом и ЭВМ - языка программирования.

Стремление программистов оперировать не цифрами, а символами, привело к созданию мнемонического языка программирования, который называют ассемблером, мнемокодом, автокодом. Этот язык имеет определенный синтаксис записи программ, в котором, в частности, цифровой код операции заменен мнемоническим кодом. Например, команда сложения записывается в виде AR 1,2 и означает сложение (Addition) типа регистр-регистр (Register) для регистров 1 и 2. Теперь программа имеет более удобочитаемую форму, но ее не понимает ЭВМ. Поэтому понадобилось создать специальную программу транслятор, который преобразует программу с языка ассемблера на МЯ. Эта проблема потребовала, в свою очередь, глубоких научных исследований и разработки различных теорий, например теорию формальных языков, которые легли в основу создания трансляторов. Практически любой класс ЭВМ имеет свой язык ассемблера. На сегодняшний день язык ассемблера используется для создания системных программ, использующих специфические аппаратные возможности данного класса ЭВМ.

Следующий этап характеризуется созданием языков высокого уровня (ЯВУ). Эти языки являются универсальными (на них можно создавать любые прикладные программы) и алгоритмически полными, имеют более широкий спектр типов данных и операций, поддерживают технологии программирования. На этих языках создается неисчислимое множество различных прикладных программ.

Принципиальными отличиями ЯВУ от языков низкого уровня являются:

· использование переменных;

· возможность записи сложных выражений;

· расширяемость типов данных за счет конструирования новых типов из базовых;

· расширяемость набора операций за счет подключения библиотек подпрограмм;
· слабая зависимость от типа ЭВМ.

С усложнением ЯП усложняются и трансляторы для них. Теперь в набор инструментов программиста, кроме транслятора, входит текстовый редактор для ввода текста программ, отладчик для устранения ошибок, библиотекарь для создания библиотек программных модулей и множество других служебных программ. Все вместе это называется системой программирования. Наиболее яркими представителями ЯВУ являются FORTRAN, PL/1, Pascal, C, Basic, Ada.

Может возникнуть вопрос: почему создано столько различных языков одного класса? Почему нельзя создать один язык на все случаи жизни? Ответ на этот вопрос может быть таким же, как и на вопрос о различных языках народов мира: так уж получилось. Каждый из разработчиков ЯВУ стремился создать самый лучший и самый универсальный язык, который позволял бы быстро получать самые эффективные, надежные и безошибочные программы. Однако в процессе этого поиска выяснилось, что дело не в самом языке, а в технологии его использования. Поэтому дальнейшее развитие языков стало определяться новыми технологиями программирования.
Одновременно с развитием универсальных ЯВУ стали развиваться проблемно-ориентированные ЯП, которые решали экономические задачи (COBOL), задачи реального времени (Modula-2, Ada), символьной обработки (Snobol), моделирования (GPSS, Simula, SmallTalk), численно-аналитические задачи (Analitic) и другие. Эти специализированные языки позволяли более адекватно описывать объекты и явления реального мира, приближая язык программирования к языку специалиста в проблемной области.
Другим направлением развития ЯП является создание языков сверхвысокого уровня (ЯСВУ). На языке высокого уровня программист задает процедуру (алгоритм) получения результата по известным исходным данным, поэтому они называются процедурными ЯП. На ЯСВУ программист задает отношения между объектами в программе, например систему линейных уравнений, и определяет, что нужно найти, но не задает как получить результат. Такие языки еще называют непроцедурными, т.к. сама процедура поиска решения встроена в язык (в его интерпретатор). Такие языки используются, например, для решения задач искусственного интеллекта (Lisp, Prolog) и позволяют моделировать мыслительную деятельность человека в процессе поиска решений.

К непроцедурным языкам относят и языки запросов систем управления базами данных

(QBE, SQL).

Классификация ЯП

Исходя из вышесказанного, ЯП можно классифицировать по следующим признакам.

1. По степени ориентации на специфические возможности ЭВМ ЯП делятся на:

· машинно-зависимые;

· машинно-независимые.

К машинно-зависимым ЯП относятся машинные языки, ассемблеры и автокоды, которые используются в системном программировании. Программа на машинно-зависимом ЯП может выполняться только на ЭВМ данного типа. Программа на машинно-независимом ЯП после трансляции на машинный язык становится машинно-зависимой. Этот признак ЯП определяет мобильность получаемых программ (возможность переноса на ЭВМ другого типа).

2. По степени детализации алгоритма получения результата ЯП делятся на:
· языки низкого уровня;
· языки высокого уровня;
· языки сверхвысокого уровня.
3. По степени ориентации на решение определенного класса задач:
· проблемно-ориентированные;
· универсальные.

4. По возможности дополнения новыми типами данных и операциями:

· расширяемые;
· нерасширяемые.

5. По возможности управления реальными объектами и процессами:
· языки систем реального времени;
· языки систем условного времени.

6. По способу получения результата:
· процедурные;
· непроцедурные.

7. По типу решаемых задач:
· языки системного программирования;
· языки прикладного программирования.

8. Непроцедурные языки по типу встроенной процедуры поиска решений делятся на:
· реляционные;
· функциональные;
· логические.

Рассмотренная схема классификации позволяет каждому ЯП присвоить один из признаков каждого класса.

Рассмотренная схема классификации ЯП позволяет сделать вывод о том, что ЯП обладают определенной специализацией. Поэтому рассмотрим тенденции развития классов ЯП.

Языки системного программирования, на которых создаются операционные системы, трансляторы и другие системные программы, развиваются в направлении повышения их уровня и независимости от ЭВМ. На сегодняшний день почти 90% системного программного обеспечения создается не на языке ассемблера, а на языке С. Например, операционная система Unix практически полностью написана на С. Язык С позволяет получать программы, сравнимые по своей эффективности с программами, написанными на языке ассемблера. Правда, объем программ получается больше, но зато эффективность их создания гораздо выше.

Машинная независимость достигается использованием стандарта языка, поддерживаемого всеми разработчиками трансляторов, и использованием так называемых кросс-систем для эквивалентного преобразования программ с одного языка низкого уровня на другой.

Другим направлением является повышение уровня самого машинного языка. Например, известны Lisp-машины, в которых машинным языком является язык Lisp (реализован аппаратно). Другим примером являются ЭВМ 5-го поколения с машинным языком искусственного интеллекта Prolog.

ЯВУ развиваются в направлении поддержки технологий программирования, обеспечения низкоуровневых операций (уровня ассемблера), обеспечения новых информационных технологий (НИТ) и независимости от среды реализации. Следует сказать, что по своим возможностям ЯВУ постепенно сближаются и программисту на С все труднее становится спорить о преимуществах языка С с программистом, работающим на языке Basic.

Тотальный бум переживает технология объектно-ориентированного программирования (ООП): практически все современные ЯВУ поддерживают ООП. Да и все современные программные системы построены на принципах ООП, и сегодня каждый программирующий студент знает, что такое инкапсуляция, наследование и полиморфизм. Для обозначения факта поддержки ООП языки получают приставку Object (например, ObjectPascal) или другие (например, C++).

Windows, сети ЭВМ, серверы, базы данных и Internet, как основа НИТ, оказывают сильнейшее влияние на современные ЯП. Разработчики ЯП просто обязаны включать в языки средства поддержки НИТ, чтобы привлечь программистов на свою сторону. Для поддержки Windows создаются системы визуального программирования с приставкой Visual, например Visual C++, Visual Basic и др. Для работы с БД, сетями и Internet в ЯП включаются специальные внутренние или внешние средства.

Стремление к созданию программ, независимых от типа ЭВМ и операционной системы, привело к созданию языка Java. Основная задача Java - обеспечить выполнение программ, распространяемых через Web-страницы Internet, на любой рабочей станции.

Кроме того, Java поддерживает все средства НИТ и в ближайшее время, очевидно, станет самым популярным ЯП.

Популярность языков искусственного интеллекта за последние 10 лет, к сожалению, заметно упала. Это связано, прежде всего, с психологическими проблемами, которые испытывают программисты при использовании этих языков. Например, в мощнейшем языке Lisp программа имеет очень сложную для понимания списочную структуру и небольшой по объему проект очень быстро выходит из-под контроля. В языке Prolog программист должен точно знать логику работы встроенной машины логического вывода, а работа программы зависит от структуры и содержимого базы знаний (БЗ). Если с проектированием программы и структуры БЗ программист справляется, то для заполнения БЗ он должен быть экспертом в предметной области либо тесно контактировать с экспертом и извлекать из него знания, а то и другое является сложной задачей.

Поэтому необходимы дополнительные обеспечивающие средства для возврата популярности этих языков.

Заключение. Изучение вопросов эволюции ЯП призвано облегчить программисту выбор языка для решения определенных задач. Однако следует осознавать, что не все мы полиглоты и не нужно изучать все существующие ЯП - достаточно изучать по одному языку каждого класса по мере необходимости, так как в процессе эволюции все языки одного класса сближаются. И помните главное: лучший язык тот, который знаешь в совершенстве.

Это всего лишь попытка «встать на цыпочки и осмотреться»… *Первые универсальные языки.*

Обратимся к истокам развития вычислительной техники. Вспомним самые первые компьютеры и программы для них. Это была эра программирования непосредственно в машинных кодах, а основным носителем информации были перфокарты и перфоленты. Программисты обязаны были знать архитектуру машины досконально. Программы были достаточно простыми, что обуславливалось, во-первых, весьма ограниченными возможностями этих машин, и, во-вторых, большой сложностью разработки и, главное, отладки программ непосредственно на машинном языке. Вместе с тем такой способ разработки давал программисту просто невероятную власть над системой. Становилось

возможным использование таких хитроумных алгоритмов и способов организации программ, какие и не снились современным разработчикам. Например, могла применяться (и применялась!) такая возможность, как самомодифицирующийся код. Знание двоичного представления команд позволяло иногда не хранить некоторые данные отдельно, а встраивать их в код как команды. И это далеко не полный список приемов, владение хотя бы одним из которых сейчас сразу же продвигает вас до уровня «гуру» экстра-класса.

Ассемблер

Первым значительным шагом представляется переход к языку ассемблера (позволим себе маленькое лирическое отступление: английское название assembly language, или assembler, на русский переводят именно тем термином, который был использован выше. При этом у новичка создается впечатление, что язык назван в честь некоего человека по имени ассемблер. Достаточно забавная ситуация, не правда ли?). Не очень заметный, казалось бы, шаг - переход к символическому кодированию машинных команд - имел на самом деле огромное значение. Программисту не надо было больше вникать в хитроумные способы кодирования команд на аппаратном уровне. Более того, зачастую одинаковые по сути команды кодировались совершенно различным образом в зависимости от своих параметров (широко известный пример из мира современных компьютеров - это кодирование инструкции mov в процессорах Intel: существует несколько совершенно по-разному кодируемых вариантов команды; выбор того или иного варианта зависит от операндов, хотя суть выполняемой операции неизменна: поместить содержимое (или значение) второго операнда в первый). Появилась также возможность использования макросов и меток, что также упрощало создание, модификацию и отладку программ. Появилось даже некое подобие переносимости - существовала возможность разработки целого семейства машин со сходной системой команд и некоего общего ассемблера для них, при этом не было нужды обеспечивать двоичную совместимость.

Вместе с тем, переход к новому языку таил в себе и некоторые отрицательные (по крайней мере, на первый взгляд) стороны. Становилось почти невозможным

использование всяческих хитроумных приемов сродни тем, что упомянуты выше. Кроме того, здесь впервые в истории развития программирования появились два представления программы: в исходных текстах и в откомпилированном виде. Сначала, пока ассемблеры только транслировали мнемоники в машинные коды, одно легко переводилось в другое и обратно, но затем, по мере появления таких возможностей, как метки и макросы, дизассемблирование становилось все более и более трудным делом. К концу ассемблерной эры возможность автоматической трансляции в обе стороны была утеряна окончательно. В связи с этим было разработано большой количество специальных программ-дизассемблеров, осуществляющих обратное преобразования, однако в большинстве случаев они с трудом могут разделить код и данные. Кроме того, вся логическая информация (имена переменных, меток и т.п.) теряется безвозвратно. В случае же задачи о декомпиляции языков высокого уровня примеры удовлетворительного решения проблемы и вовсе единичны.

Фортран

В 1954 году в недрах корпорации IBM группой разработчиков во главе с Джоном Бэкусом (John Backus) был создан язык программирования Fortran.

Значение этого события трудно переоценить. Это первый язык программирования высокого уровня. Впервые программист мог по-настоящему абстрагироваться от особенностей машинной архитектуры. Ключевой идеей, отличающей новый язык от ассемблера, была концепция подпрограмм. Напомним, что это современные компьютеры поддерживают подпрограммы на аппаратном уровне, предоставляя соответствующие команды и структуры данных (стек) прямо на уровне ассемблера, в 1954 же году это было совершенно не так. Поэтому компиляция Fortran'а была процессом отнюдь не тривиальным. Кроме того, синтаксическая структура языка была достаточно сложна для машинной обработки в первую очередь из-за того, что пробелы как синтаксические единицы вообще не использовались. Это порождало массу возможностей для скрытых ошибок, таких, например:

Автор: Александр
26.08.2014 14:51

В Фортране следующая конструкция описывает «цикл for до метки 10 при изменении индекса от 1 до 100» DO 10 I=1,100
Если же здесь заменить запятую на точку, то получится оператор присваивания:
DO10I = 1.100

Говорят, что такая ошибка заставила ракету взорваться во время старта!

Язык Фортран использовался (и используется по сей день) для научных вычислений. Он страдает от отсутствия многих привычных языковых конструкций и атрибутов, компилятор практически никак не проверяет синтаксически правильную программу с точки зрения семантической корректности (соответствие типов и проч.). В нем нет поддержки современных способов структурирования кода и данных. Это осознавали и сами разработчики. По признанию самого Бэкуса, перед ними стояла задача скорее разработки компилятора, чем языка. Понимание самостоятельного значения языков программирования пришло позже.

Появление Фортрана было встречено еще более яростной критикой, чем внедрение ассемблера. Программистов пугало снижение эффективности программ за счет использования промежуточного звена в виде компилятора. И эти опасения имели под собой основания: действительно, хороший программист, скорее всего, при решении какой-либо небольшой задачи вручную напишет код, работающий быстрее, чем код, полученный как результат компиляции. Через некоторое время пришло понимание того, что реализация больших проектов невозможна без применения языков высокого уровня. Мощность вычислительных машин росла, и с тем падением эффективности, которое раньше считалось угрожающим, стало возможным смириться. Преимущества же языков высокого уровня стали настолько очевидными, что побудили разработчиков к созданию новых языков, все более и более совершенных.

Cobol

Автор: Александр
26.08.2014 14:51

В 1960 году был создан язык программирования Cobol.

Он задумывался как язык для создания коммерческих приложений, и он стал таковым. На Коболе написаны тысячи прикладных коммерческих систем. Отличительной особенностью языка является возможность эффективной работы с большими массивами данных, что характерно именно коммерческих приложений. Популярность Кобола столь высока, что даже сейчас, при всех его недостатках (по структуре и замыслу Кобол во многом напоминает Фортран) появляются новые его диалекты и реализации. Так недавно появилась реализация Кобола, совместимая с Microsoft.NET, что потребовало, вероятно, внесения в язык некоторых черт объектно-ориентированного языка.

PL/1

В 1964 году все та же корпорация IBM создала язык PL/1, который был призван заменить Cobol и Fortran в большинстве приложений. Язык обладал исключительным богатством синтаксических конструкций. В нем впервые появилась обработка исключительных ситуаций и поддержка параллелизма. Надо заметить, что синтаксическая структура языка была крайне сложной. Пробелы уже использовались как синтаксические разделители, но ключевые слова не были зарезервированы. В частности, следующая строка - это вполне нормальный оператор на PL/1:

IF ELSE=THEN THEN THEN; ELSE ELSE

В силу таких особенностей разработка компилятора для PL/1 была исключительно сложным делом. Язык так и не стал популярен вне мира IBM.

Автор: Александр
26.08.2014 14:51

BASIC

В 1963 году в Дартмурском колледже (Dartmouth College) был создан язык программирования BASIC (Beginners' All-Purpose Symbolic Instruction Code - многоцелевой язык символических инструкций для начинающих). Язык задумывался в первую очередь как средство обучения и как первый изучаемый язык программирования. Он предполагался легко интерпретируемым и компилируемым. Надо сказать, что BASIC действительно стал языком, на котором учатся программировать (по крайней мере, так было еще несколько лет назад; сейчас эта роль отходит к Pascal). Было создано несколько мощных реализаций BASIC, поддерживающих самые современные концепции программирования (ярчайший пример - Microsoft Visual Basic).

Algol

В 1960 году командой во главе с Петером Науром (Peter Naur) был создан язык программирования Algol. Этот язык дал начало целому семейству Алгол-подобных языков (важнейший представитель - Pascal). В 1968 году появилась новая версия языка. Она не нашла столь широкого практического применения, как первая версия, но была весьма популярна в кругах теоретиков. Язык был достаточно интересен, так как обладал многими уникальными на так момент характеристиками.

Автор: Александр
26.08.2014 14:51

Дальнейшее развитие языков программирования

Создание каждого из вышеупомянутых языков (за исключением, может быть, Algol'a) было вызвано некоторыми практическими требованиями. Эти языки послужили фундаментом для более поздних разработок. Все они представляют одну и ту же парадигму программирования. Следующие языки пошли существенно дальше в своем развитии, в сторону более глубокого абстрагирования.

Сведения о более поздних языках я буду приводить в виде описания семейств языков. Это позволит лучше проследить взаимосвязи между отдельными языками

Pascal-подобные языки

В 1970 году Никлаусом Виртом был создал язык программирования Pascal. Язык замечателен тем, что это первый широко распространенный язык для структурного программирования (первым, строго говоря, был Алгол, но он не получил столь широкого распространения). Впервые оператор безусловного перехода перестал играть основополагающую роль при управлении порядком выполнения операторов. В этом языке также внедрена строгая проверка типов, что позволило выявлять многие ошибки на этапе компиляции.

Отрицательной чертой языка было отсутствие в нем средств для разбиения программы на модули. Вирт осознавал это и разработал язык Modula-2 (1978), в котором идея модуля стала одной из ключевых концепций языка. В 1988 году появилась Modula-3, в которую были добавлены объектно-ориентированные черты. Логическим продолжением Pascal и Modula являются язык Oberon и Oberon-2. Они характеризуются движением в сторону объектно- и компонентно- ориентированности.

Автор: Александр
26.08.2014 14:51

C-подобные языки

В 1972 году Керниганом и Ритчи был создан язык программирования C. Он создавался как язык для разработки операционной системы UNIX. C часто называют «переносимым ассемблером», имея в виду то, что он позволяет работать с данными практически так же эффективно, как на ассемблере, предоставляя при этом структурированные управляющие конструкции и абстракции высокого уровня (структуры и массивы). Именно с этим связана его огромная популярность и поныне. И именно это является его ахиллесовой пятой. Компилятор C очень слабо контролирует типы, поэтому очень легко написать внешне совершенно правильную, но логически ошибочную программу.

В 1986 году Бьярн Страуструп создал первую версию языка C++, добавив в язык C объектно-ориентированные черты, взятые из Simula (см. ниже), и исправив некоторые ошибки и неудачные решения языка. C++ продолжает совершенствоваться и в настоящее время, так в 1998 году вышла новая (третья) версия стандарта, содержащая в себе некоторые довольно существенные изменения. Язык стал основой для разработки современных больших и сложных проектов. У него имеются, однако же, и слабые стороны, вытекающие из требований эффективности.

В 1995 году в корпорации Sun Microsystems Кеном Арнольдом и Джеймсом Гослингом был создан язык Java. Он наследовал синтаксис C и C++ и был избавлен от некоторых неприятных черт последнего. Отличительной особенностью языка является компиляция в код некоей абстрактной машины, для которой затем пишется эмулятор (Java Virtual Machine) для реальных систем. Кроме того, в Java нет указателей и множественного наследования, что сильно повышает надежность программирования.

В 1999–2000 годах в корпорации Microsoft был создан язык C#. Он в достаточной степени схож с Java (и задумывался как альтернатива последнему), но имеет и

отличительные особенности. Ориентирован, в основном, на разработку многокомпонентных Интернет-приложений.

Языки Ada и Ada 95

В 1983 году под эгидой Министерства Обороны США был создан язык Ada. Язык замечателен тем, что очень много ошибок может быть выявлено на этапе компиляции. Кроме того, поддерживаются многие аспекты программирования, которые часто отдаются на откуп операционной системе (параллелизм, обработка исключений). В 1995 году был принят стандарт языка Ada 95, который развивает предыдущую версию, добавляя в нее объекно- ориентированность и исправляя некоторые неточности. Оба этих языка не получили широкого распространения вне военных и прочих крупномасштабных проектов (авиация, железнодорожные перевозки). Основной причиной является сложность освоения языка и достаточно громоздкий синтаксис (значительно более громоздкий, чем Pascal).

Языки обработки данных

Автор: Александр
26.08.2014 14:51

Все вышеперечисленные языки являются языками общего назначения в том смысле, что они не ориентированы и не оптимизированы под использование каких-либо специфических структур данных или на применение в каких-либо специфических областях. Было разработано большое количество языков, ориентированных на достаточно специфические применения. Ниже приведен краткий обзор таких языков.

APL

В 1957 году была предпринята попытка создания языка для описания математической обработки данных. Язык был назван APL (Application Programming Language). Его отличительной особенностью было использование математических символов (что затрудняло применение на текстовых терминалах; появление графических интерфейсов сняло эту проблему) и очень мощный синтаксис, который позволял производить множество нетривиальных операций прямо над сложными объектами, не прибегая к разбиению их на компоненты. Широкому применению помешало, как уже отмечалось, использование нестандартных символов как элементов синтаксиса.

Snobol и Icon

В 1962 году появился язык Snobol (а в 1974 - его преемник Icon), предназначенный для обработки строк. Синтаксис Icon напоминает C и Pascal одновременно. Отличие заключается в наличии мощных встроенных функций работы со строками и связанная с

этими функциями особая семантика. Современным аналогом Icon и Snobol является Perl - язык обработки строк и текстов, в который добавлены некоторые объектно-ориентированные возможности. Считается очень практичным языком, однако ему недостает элегантности.

SETL

В 1969 году был создан язык SETL - язык для описания операций над множествами. Основной структурой данных в языке является множество, а операции аналогичны математическим операциям над множествами. Полезен при написании программ, имеющих дело со сложными абстрактными объектами.

Lisp и ему подобные языки

В 1958 году появился язык Lisp - язык для обработки списков. Получил достаточно широкое распространение в системах искусственного интеллекта. Имеет несколько потомков: Planner (1967), Scheme (1975), Common Lisp (1984). Многие его черты были унаследованы современными языками функционального программирования.

Автор: Александр
26.08.2014 14:51

Скриптовые языки

В последнее время в связи развитием Интернет-технологий, широким распространением высокопроизводительных компьютеров и рядом других факторов получили распространение так называемые скриптовые языки. Эта языки первоначально ориентировались на использование в качестве внутренних управляющих языков во всякого рода сложных системах. Многие из них, однако же, вышли за пределы сферы своего изначального применения и используются ныне в совсем иных областях. Характерными особенностями данных языков являются, во-первых, их интерпретируемость (компиляция либо невозможна, либо нежелательна), во-вторых, простота синтаксиса, а в-третьих, легкая расширяемость. Таким образом, они идеально подходят для использования в часто изменяемых программах, очень небольших программах или в случаях, когда для выполнения операторов языка затрачивается время, несопоставимое со временем их разбора. Было создано достаточно большое количество таких языков, перечислим лишь основные и наиболее часто используемые.

JavaScript

Язык был создан в компании Netscape Communications в качестве языка для описания сложного поведения веб-страниц. Первоначально назывался LiveScript, причиной смены названия получили маркетинговые соображения. Интерпретируется браузером во время отображения веб-страницы. По синтаксису схож с Java и (отдаленно) с C/C++. Имеет возможность использовать встроенную в браузер объектную функциональность, однако подлинно объектно-ориентированным языком не является.

VBScript

Язык был создан в корпорации Microsoft во многом в качестве альтернативы JavaScript. Имеет схожую область применения. Синтаксически схож с языком Visual Basic (и является усеченной версией последнего). Так же, как и JacaScript, исполняется браузером при отображении веб-страниц и имеет ту же степень объектно-ориентированности.

Perl

Язык создавался в помощь системному администратору операционной системы Unix для обработки различного рода текстов и выделения нужной информации. Развился до мощного средства работы с текстами. Является интерпретируемым языком и реализован практически на всех существующих платформах. Применяется при обработке текстов, а также для динамической генерации веб-страниц на веб-серверах.

Python

Автор: Александр
26.08.2014 14:51

Интерпретируемый объектно-ориентированный язык программирования. По структуре и области применения близок к Perl, однако менее распространен и более строг и логичен. Имеются реализации для большинства существующих платформ.

Объектно-ориентированные языки

Объектно-ориентированный подход, пришедший на смену структурному, впервые появился отнюдь не в C++, как полагают некоторые. Существует целая череда чистых объектно-ориентированных языков, без сведений о которых наш обзор был бы неполным.

Simula

Первым объектно-ориентрованным языком был язык Simula (1967). Этот язык был предназначен для моделирования различных объектов и процессов, и объектно-ориентированные черты появились в нем именно для описания свойств модельных объектов.

Автор: Александр
26.08.2014 14:51

Smalltalk

Популярность объектно-ориентированному программированию принес язык Smalltalk, созданный в 1972 году. Язык предназначался для проектирования сложных графических интерфейсов и был первым по-настоящему объектно-ориентированным языком. В нем классы и объекты - это единственные конструкции программирования. Большим недостатком Smalltalk являются большие требования к памяти и низкая производительность полученных программ. Это связано с не очень удачной реализацией объектно-ориентированных особенностей. Популярность языков C++ и Ada 95 связана именно с тем, что объектно- ориентированность реализована без существенного снижения производительности.

Eiffel

Существует язык с очень хорошей реализацией объектно- ориентированности, не являющийся надстройкой ни над каким другим языком. Это язык Eiffel (1986). Являясь чистым языком объектно-ориентированного программирования, он, кроме того, повышает надежность программы путем использования «контрольных утверждений».

Языки параллельного программирования

Автор: Александр
26.08.2014 14:51

Большинство компьютерных архитектур и языков программирования ориентированы на последовательное выполнение операторов программы. В настоящее время, однако же, существуют программно-аппаратные комплексы, позволяющие организовать параллельное выполнение различных частей одного и того же вычислительного процесса. Для программирования таких систем необходима специальная поддержка со стороны средств программирования, в частности, языков программирования. Некоторые языки общего назначения содержат в себе элементы поддержки параллелизма, однако же программирование истинно параллельных систем требует подчас специальных приемов.

Язык Occam

Язык Occam был создан в 1982 году и предназначен для программирования транспьютеров - многопроцессорных систем распределенной обработки данных. Он описывает взаимодействие параллельных процессов в виде каналов - способов передачи информации от одного процесса к другому. Отметим особенность синтаксиса языка Щccam - в нем последовательный и параллельный порядки выполнение операторов равноправны, и их необходимо явно указывать ключевыми словами PAR и SEQ.

Модель параллельных вычислений Linda

Автор: Александр
26.08.2014 14:51

В 1985 году была предложена модель параллельных вычислений Linda. Основной ее задачей является организация взаимодействия между параллельно выполняющимися процессами. Это достигается за счет использования глобальной кортежной области (tuple space). Процесс может поместить туда кортеж с данными (то есть совокупность нескольких, возможно разнородных, данных), а другой процесс может ожидать появления в кортежной области некоторого кортежа и, после его появления, прочитать кортеж с возможным последующим его удалением. Заметим, что процесс может, например, поместить кортеж в область и завершиться, а другой процесс может через некоторое время воспользоваться этим кортежем. Таким образом обеспечивается возможность асинхронного взаимодействия. Очевидно, что при помощи такой модели может быть сэмулировано и синхронное взаимодействие. Linda - это модель параллельных вычислений, она может быть добавлена в любой язык программирования. Существуют достаточно эффективные реализации Linda, обходящие проблему существования глобальной кортежной области с потенциально неограниченным объемом памяти.

Неимперативные языки

Все языки, о которых шла речь ранее, имеют одно общее свойство: они императивны. Это означает, что программы на них, в конечном итоге, представляют собой пошаговое описание решения той или иной задачи. Можно попытаться описывать лишь постановку проблемы, а решать задачу поручить компилятору. Существует два основных подхода, развивающие эту идею: функциональное и логическое программирование.

Автор: Александр
26.08.2014 14:51

Функциональные языки

Основная идея, лежащая в основе функционального программирования, - это представление программы в виде математических функций (т.е. функций, значение которых определяется лишь их аргументами, а не контекстом выполнения). Оператор присваивания в таких языках не используется (или, как минимум, его использование не поощряется). Императивные возможности, как правило, имеются, но их применение обставлено серьезными ограничениями. Существуют языки с ленивой и с энергичной семантикой. Различие заключается, грубо говоря, в том, что в языках с энергичной семантикой вычисления производятся в том же месте, где они описаны, а в случае ленивой семантики вычисление производится только тогда, когда оно действительно необходимо. Первые языки имеют более эффективную реализацию, в то время как вторые - лучшую семантику.

Из языков с энергичной семантикой упомянем ML и два его современных диалекта - Standard ML (SML) и CaML. Последний имеет объектно-ориентированного потомка - Objective CaML (O'CaML).

Среди языков с ленивой семантикой наиболее распространены два: Haskell и его более простой диалект Clean.

Языки логического программирования

Автор: Александр
26.08.2014 14:51

Программы на языках логического программирования выражены как формулы математической логики, а компилятор пытается получить следствия из них.

Родоначальником большинства языков логического программирования является язык Prolog (1971). У него есть ряд потомков - Parlog (1983, ориентирован на параллельные вычисления), Delta Prolog и др. Логическое программирование, как и функциональное, - это отдельная область программирования, и за более подробными сведениями мы отсылаем читателя к специальной литературе.

Языки развиваются в сторону все большей и большей абстракции. И это сопровождается падением эффективности. Вопрос: а стоит ли этого абстракция? Ответ: стоит. Стоит, так как повышение уровня абстракции влечет за собой повышение уровня надежности программирования. С низкой эффективностью можно бороться путем создания более быстрых компьютеров. Если требования к памяти слишком высоки, можно увеличить ее объем. Это, конечно, требует времени и средств, но это решаемо. А вот с ошибками в программах можно бороться только одним способом: их надо исправлять. А еще лучше - не совершать. А еще лучше максимально затруднить их совершение. И именно на это направлены все исследования в области языков программирования. А с потерей эффективности придется смириться.