Глоссарий

Программирование

Язык программирования

Массив

Цикл

Оператор цикла

Тип данных

Переменная

Константа

Оператор присваивания

Подпрограммы (функция, процедура)

Объект

Класс

Наследование

Рекурсия

Программирование, процесс составления упорядоченной последовательности действий (программы) для ЭВМ; научная дисциплина, изучающая программы для ЭВМ и способы их составления, проверки и улучшения.

Аль-Джазари построил программируемый автомат-гуманоид в 1206 году. Одна система, задействованная в этих устройствах, использовала зажимы и кулачки, помещённые в деревянный ящик в определённых местах, которые последовательно задействовали рычаги, которые, в свою очередь, управляли ударными инструментами.

Часто первым программируемым устройством принято считать жаккардовый ткацкий станок, построенный в 1804 году Жозефом Мари Жаккаром, который произвёл революцию в ткацкой промышленности, предоставив возможность программировать узоры на тканях при помощи перфокарт.

Первое программируемое вычислительное устройство, Аналитическую машину, разработал Чарлз Бэббидж (но не смог её построить). 19 июля 1843 года графиня Ада Августа Лавлейс, дочка великого английского поэта Джорджа Байрона, как принято считать, написала первую в истории человечества программу для Аналитической машины. Эта программа решала уравнение Бернулли, выражающее закон сохранения энергии движущейся жидкости. В своей первой и единственной научной работе Ада Лавлейс рассмотрела большое число вопросов. Ряд высказанных ею общих положений (принцип экономии рабочих ячеек памяти, связь рекуррентных формул с циклическими процессами вычислений) сохранили свое принципиальное значение и для современного программирования. В материалах Бэббиджа и комментариях Лавлейс намечены такие понятия, как подпрограмма и библиотека подпрограмм, модификация команд и индексный регистр, которые стали употребляться только в 1950-х годах. Однако ни одна из программ написанных Адой Лавлейс никогда так и не была запущена

Назад

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 её управлением.

Со времени создания первых программируемых машин человечество придумало более двух с половиной тысяч языков программирования. Каждый год их число увеличивается. 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.

Функция: язык программирования предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению того или иного вычислительного процесса и организации управления отдельными устройствами.

Задача: язык программирования отличается от естественных языков тем, что предназначен для передачи команд и данных от человека к компьютеру, в то время как естественные языки используются для общения людей между собой. Можно обобщить определение «языков программирования» — это способ передачи команд, приказов, чёткого руководства к действию; тогда как человеческие языки служат также для обмена информацией.

Исполнение: язык программирования может использовать специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

Назад

Массив — это переменная, в которой хранится одновременно несколько значений одинакового типа. Формальное определение массива таково: он представляет собой совокупность однотипных индексированных переменных.

Количество используемых индексов массива также может быть различным. Чаще всего применяются массивы с одним или двумя индексами, реже — с тремя, еще большее количество индексов встречается крайне редко. В VBA допускается использовать до 60 индексов. О количестве индексов массива обычно говорят как о размерности массива. Массивы с одним индексом называют одномерным, с двумя — двумерными и т.д. Массивы с большим количеством измерений могут занимать очень большие объемы памяти, так что следует быть осторожным в их применении.

Назад

Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая многократно исполняемая последовательность инструкций, организованная любым способом

Последовательность инструкций, предназначенная для многократного исполнения, называется телом цикла.

Единичное выполнение тела цикла называется итерацией.

Выражение определяющее, будет в очередной раз выполняться итерация, или цикл завершится, называется условием выхода или условием окончания цикла (либо условием продолжения в зависимости от того, как интерпретируется его истинность — как признак необходимости завершения или продолжения цикла).

Переменная, хранящая текущий номер итерации, называется счётчиком итераций цикла или просто счётчиком цикла. Цикл не обязательно содержит счётчик, счётчик не обязан быть один — условие выхода из цикла может зависеть от нескольких изменяемых в цикле переменных, а может определяться внешними условиями (например, наступлением определённого времени), в последнем случае счётчик может вообще не понадобиться.

Исполнение любого цикла включает первоначальную инициализацию переменных цикла, проверку условия выхода, исполнение тела цикла и обновление переменной цикла на каждой итерации. Кроме того, большинство языков программирования предоставляют средства для досрочного управления циклом, например, операторы завершения цикла, то есть выхода из цикла независимо от истинности условия выхода и операторы пропуска итерации.

Назад

В ТurboРascal существует три оператора цикла:

цикл с предусловием;

цикл с постусловием;

цикл с параметром.

Для всех циклов характерны следующие особенности:

значения переменных используемых в цикле, и не изменяющиеся в нем определены до входа в цикл;

вход в цикл возможен только через его начало;

выход их цикла осуществляется как в результате его естественного окончания, так и с помощью операторов перехода.

Оператор цикла с предусловием в Паскале

Оператор цикла с предусловием реализует следующую базовую конструкцию:

Формат записи:

While L do OP;

где:

While - пока не;

do – выполнить;

L – выражение логического типа;

OP – тело цикла; оператор (простой или составной).

Оператор цикла с постусловием в языке Паскаль

Оператор цикла с постусловием реализует следующую конструкцию:

Формат записи:

Repeat OP Until L;

где:

Repeat - повторять;

Until – пока не;

L – выражение логического типа;

OP – тело цикла; оператор (простой или составной).

Оператор цикла с параметром в языке Паскаль

Формат записи:

 1. For P:=Pn to Pk do OP;

 2. For P:=Pk downto Pn do OP;

где:

For - для;

to – до;

downto – уменьшая до

do – выполнить;

OP – тело цикла; оператор (простой или составной);

P - параметр цикла, переменная порядкового типа;

Pn, Pk – начальное и конечное значение параметра.

Назад

Тип данных — фундаментальное понятие теории программирования. Тип данных определяет множество значений, набор операций, которые можно применять к таким значениям, и, возможно, способ реализации хранения значений и выполнения операций. Любые данные, которыми оперируют программы, относятся к определённым типам.

Типы данных различаются начиная с нижних уровней системы. Так, например, даже в Ассемблере х86 различаются типы «целое число» и «вещественное число». Это объясняется тем, что для чисел рассматриваемых типов отводятся различные объёмы памяти, используются различные регистры микропроцессора, а для операций с ними применяются различные команды Ассемблера и различные ядра микропроцессора.

Концепция типа данных появилась в языках программирования высокого уровня как естественное отражение того факта, что обрабатываемые программой данные могут иметь различные множества допустимых значений, храниться в памяти компьютера различным образом, занимать различные объёмы памяти и обрабатываться с помощью различных команд процессора.

Назад

Переменная в императивном программировании — поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются значением этой переменной.

В других парадигмах программирования, например, в функциональной и логической, понятие переменной оказывается несколько иным. В таких языках переменная определяется как имя, с которым может быть связано значение, или даже как место (location) для хранения значения.

Область видимости и/или время существования переменной в некоторых языках задаётся классом памяти.

Если тип переменной определяется на этапе компиляции, имеет место статическая типизация, а если на этапе выполнения программы — динамическая. В последнем случае иногда говорят, что переменная не имеет типа, хотя данные, содержащиеся в ней, безусловно, относятся к определённому типу данных, но выясняется это уже во время выполнения программы.

В большинстве случаев статическая типизация позволяет уменьшить затраты ресурсов при выполнении программы, поскольку для динамической типизации требуются затраты ресурсов на выяснение типов данных их приведение в выражениях с смешанными типами. Статическая типизация позволяет перекладывать проверку типов на этапе компиляции программы. Это также упрощает обнаружение ошибок ещё на этапе разработки, когда их исправление обходится менее дорого. Тем не менее во многих случаях необходимо применение динамической типизации. Например, необходимость поддержания совместимости при переходе на новый формат представления данных (например, старая часть проекта посылает процедуре дату символьной строкой, а новые объекты используют более современный числовой тип).

Адрес поименованной ячейки памяти также может определяться как на этапе компиляции, так и во время выполнения программы. По времени создания переменные бывают статическими и динамическими. Первые создаются в момент запуска программы или подпрограммы, а вторые создаются в процессе выполнения программы.

Динамическая адресация нужна только тогда, когда количество поступающих на хранение данных заранее точно не известно. Такие данные размещают в специальных динамических структурах, тип которой выбирается в соответствии со спецификой задачи и с возможностями выбранной системы программирования.

По наличию внутренней структуры, переменные могут быть простыми или сложными (составными).

Простые переменные не имеют внутренней структуры, доступной для адресации. Последняя оговорка важна потому, что для компилятора или процессора переменная может быть сколь угодно сложной, но конкретная система (язык) программирования скрывает от программиста её внутреннюю структуру, позволяя адресоваться только «в целом».

Сложные переменные программист создаёт для хранения данных, имеющих внутреннюю структуру. Соответственно, есть возможность обратиться напрямую к любому элементу. Самыми характерными примерами сложных типов являются массив (все элементы однотипные) и запись (элементы могут иметь разный тип).

Назад

Константа в программировании — способ адресования данных, изменение которых рассматриваемой программой запрещено.

Использование констант, особенно, именованных — мощный инструмент, повышающий надёжность и безошибочность программ.

Сначала константы просто вписывались в текст программ в качестве аргументов операторов.

Например, mvi A, 0 , где 0 является числовой константой.

Затем ассемблерные компиляторы научили понимать именованные константы, серьёзно упростившие процесс отладки и сопровождения программ.

Исчезла необходимость помнить конкретные числа — разумно придуманные имена запоминаются легче.

Ошибки в именах в большом числе случаев выявляются автоматически компилятором (кроме ситуаций, когда ошибочно использовано имя другой константы аналогичного типа).

Процесс внесения изменений стал быстр и безошибочен — значение константы присутствует в программе всего в одном месте.

Ассемблеры не умели защищать от изменения косвенно адресуемые области памяти. Но такая возможность появилась в языках программирования высокого уровня. Те из них, которые поддерживают использование констант любых статических типов, безусловно, позволяют писать гораздо более надёжный и долговечный код.

Разумеется, преимущества именованных констант реализуются только при соответствующем стиле работы программиста, начиная с самых ранних этапов разработки программы.

Методологически очень важн? понимание именованной константы, как функции без аргументов. Такой взгляд позволяет, в соответствии с принципом Оккама, разумно сократить число сущностей, лежащих в основе программирования. Естественно, что истинная функция без аргументов не может возвращать других значений, кроме того, которое заложено в неё при разработке (редактировании кода).

Назад

Присваивание — механизм в программировании, позволяющий динамически изменять связи объектов данных (как правило, переменных) с их значениями. Строго говоря, изменение значений является побочным эффектом операции присвоения, и во многих современных языках программирования сама операция также возвращает некоторый результат (как правило, копию присвоенного значения). На физическом уровне результат операции присвоения состоит в проведении записи и перезаписи ячеек памяти или регистров процессора.

Присваивание является одной из центральных конструкций в императивных языках программирования, эффективно и просто реализуется на фон-неймановской архитектуре, которая является основой современных компьютеров.

В логическом программировании принят другой, алгебраический подход. Обычного («разрушающего») присвоения здесь нет. Существуют только неизвестные, которые ещё не вычислены, и соответствующие идентификаторы для обозначения этих неизвестных. Программа только определяет их значения, сами они постоянны. Конечно, в реализации программа производит запись в память, но языки программирования этого не отражают, давая программисту возможность работать с идентификаторами постоянных значений, а не с переменными.

В чистом функциональном программировании не используются переменные, и явный оператор присваивания не нужен.

Общий синтаксис простого присваивания выглядит следующим образом:

<выражение слева> <оператор присваивания > <выражение справа>

«Выражение слева» должно после вычисления привести к местоположению объекта данных, к целевой переменной, идентификатору ячейки памяти, в которую будет производиться запись. Такие ссылки называются «левосторонними значениями». Типичные примеры левостороннего значения — имя переменной (x), путь к переменной в пространстве имён и библиотеках, путь к массиву с выражением на месте индекса, но ниже в данной статье приведены и более сложные варианты.

«Выражение справа» должно обозначать тем или иным способом ту величину, которая будет присвоена объекту данных. Таким образом, даже если справа стоит имя той же переменной, что и слева, интерпретируется оно иначе — такие ссылки называются «правосторонними значениями». Дальнейшие ограничения на выражение накладывает используемый язык: так, в статически типизированных языках оно должно иметь тот же тип, что и целевая переменная, либо тип, приводимый к нему; в некоторых языках (например, Си или Python) в выражение может входить также другая операция присваивания (a=b=c).

В роли оператора присваивания в языках программирования чаще всего выступают =, := или ?. Но специальный синтаксис может и не вводится — например, в Tcl.

Назад

Подпрограмма — поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы. В языках программирования для оформления и использования подпрограмм существуют специальные синтаксические средства.

Подпрограммы изначально появились как средство оптимизации программ по объёму занимаемой памяти — они позволили не повторять в программе идентичные блоки кода, а описывать их однократно и вызывать по мере необходимости. К настоящему времени данная функция подпрограмм стала вспомогательной, главное их назначение — структуризация программы с целью удобства её понимания и сопровождения.

Выделение набора действий в подпрограмму и вызов её по мере необходимости позволяет логически выделить целостную подзадачу, имеющую типовое решение. Такое действие имеет ещё одно (помимо экономии памяти) преимущество перед повторением однотипных действий: любое изменение (исправление ошибки, оптимизация, расширение функциональности), сделанное в подпрограмме, автоматически отражается на всех её вызовах, в то время как при дублировании каждое изменение необходимо вносить в каждое вхождение изменяемого кода.

Даже в тех случаях, когда в подпрограмму выделяется однократно производимый набор действий, это оправдано, так как позволяет сократить размеры целостных блоков кода, составляющих программу, то есть сделать программу более понятной и обозримой...

В языках программирования высокого уровня используется два типа подпрограмм: процедуры и функции.

Функция — это подпрограмма специального вида, которая, кроме получения параметров, выполнения действий и передачи результатов работы через параметры имеет ещё одну возможность — она может возвращать результат. Вызов функции является, с точки зрения языка программирования, выражением, он может использоваться в других выражениях или в качестве правой части присваивания.

Процедура — это независимая именованная часть программы, которую после однократного описания можно многократно вызвать по имени из последующих частей программы для выполнения определенных действия.

Назад

Объект в программировании — некоторая сущность в виртуальном пространстве, обладающая определённым состоянием и поведением, имеющая заданные значения свойств (атрибутов) и операций над ними (методов). Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Термины «экземпляр класса» и «объект» взаимозаменяемы.

Объект, наряду с понятием класс, является важным понятием объектно-ориентированного подхода. Объекты обладают свойствами наследования, инкапсуляции и полиморфизма.

Термин объект в программном обеспечении впервые был введен в языке Simula и применялся для моделирования реальности.

Назад

В объектно-ориентированной программе с применением классов каждый объект является «экземпляром» некоторого конкретного класса, и других объектов не предусмотрено. То есть «экземпляр класса» в данном случае означает не «пример некоторого класса» или «отдельно взятый класс», а «объект, типом которого является какой-то класс». При этом в разных языках программирования допускается либо не допускается существование еще каких-то типов данных, экземпляры которых не являются объектами (то есть язык определяет, являются ли объектами такие вещи, как числа, массивы и указатели, или не являются, и, соответственно, есть ли такие классы как «число», «массив» или «указатель», экземплярами которых были бы каждое конкретное число, массив или указатель).

Например, абстрактный тип данных «строка текста» может быть оформлен в виде класса, и тогда все строки текста в программе будут являться объектами — экземплярами класса «строка текста».

При использовании классов все элементы кода программы, такие как переменные, константы, методы, процедуры и функции, могут принадлежать (а во многих языках обязаны принадлежать) тому или иному классу. Сам класс в итоге определяется как список своих членов, а именно полей (свойств) и методов/функций/процедур. В зависимости от языка программирования к этому списку могут добавиться константы, атрибуты и внешние определения.

Как и структуры, классы могут задавать поля — то есть переменные, принадлежащие либо непосредственно самому классу (статические), либо экземплярам класса (обычные). Статические поля существуют в одном экземпляре на всю программу (или, в более сложном варианте, — в одном экземпляре на процесс или на поток/нить). Обычные поля создаются по одной копии для каждого конкретного объекта — экземпляра класса. Например, общее количество строк текста, созданных в программе за время её работы, будет являться статическим полем класса «строка текста». А конкретный массив символов строки будет являться обычным полем экземпляра класса «строка текста», так же как переменная «фамилия», имеющая тип «строка текста», будет являться обычным полем каждого конкретного экземпляра класса «человек».

В ООП при использовании классов весь исполняемый код программы (алгоритмы) будет оформляться в виде так называемых «методов», «функций» или «процедур», что соответствует обычному структурному программированию, однако теперь они могут (а во многих языках обязаны) принадлежать тому или иному классу. Например, по возможности, класс «строка текста» будет содержать все основные методы/функции/процедуры, предназначенные для работы со строкой текста, такие как поиск в строке, вырезание части строки и т.д.

Как и поля, код в виде методов/функций/процедур, принадлежащих классу, может быть отнесен либо к самому классу, либо к экземплярам класса. Метод, принадлежащий классу и соотнесенный с классом (статический метод) может быть вызван сам по себе и имеет доступ к статическим переменным класса. Метод, соотнесенный с экземпляром класса (обычный метод), может быть вызван только у самого объекта, и имеет доступ как к статическим полям класса, так и к обычным полям конкретного объекта (при вызове этот объект передастся скрытым параметром метода). Например, общее количество созданных строк можно узнать из любого места программы, но длину конкретной строки можно узнать только указав, тем или иным образом, длину какой строки будем мерить.

Сам класс не является объектом, однако, в зависимости от языка программирования и платформы, программисту могут быть доступны те или иные объекты, позволяющие получить информацию о классе — название, список членов, объем памяти, занимаемой классом или отдельным экземпляром.

Назад

Наследование — механизм объектно-ориентированного программирования, позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Типы наследования

Простое наследование

Класс, от которого произошло наследование, называется базовым или родительским. Классы, которые произошли от базового, называются потомками, наследниками или производными классами.

В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т.д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Множественное наследование

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Из других языков, предоставляющих эту возможность, можно отметить Python и Эйфель. Множественное наследование поддерживается в языке UML.

Множественное наследование — потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках. В языках, которые позиционируются как наследники C++ (Java, C# и др.), от множественного наследования было решено отказаться в пользу интерфейсов. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость все-таки возникла, то, для разрешения конфликтов использования наследованных методов с одинаковыми именами, возможно, например, применить операцию расширения видимости — «::» — для вызова конкретного метода конкретного родителя.

Назад

В программировании рекурсия — вызов функции (процедуры) из неё же самой,

непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция вызывает функцию , а функция — функцию . Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

Преимущество рекурсивного определения объекта заключается в том, что такое конечное определение теоретически способно описывать бесконечно большое число объектов. С помощью рекурсивной программы же возможно описать бесконечное вычисление, причём без явных повторений частей программы.

Реализация рекурсивных вызовов функций в практически применяемых языках и средах программирования, как правило, опирается на механизм стека вызов — адрес возврата и локальные переменные функции записываются в стек, благодаря чему каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счёт этого работает корректно. Оборотной стороной этого довольно простого по структуре механизма является то, что на каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, и при чрезмерно большой глубине рекурсии может наступить переполнение стека вызовов. Вследствие этого, обычно рекомендуется избегать рекурсивных программ, которые приводят (или в некоторых условиях могут приводить) к слишком большой глубине рекурсии.

Имеется специальный тип рекурсии, называемый «хвостовой рекурсией». Интерпретаторы и компиляторы функциональных языков программирования, поддерживающие оптимизацию кода (исходного или исполняемого), автоматически преобразуют хвостовую рекурсию к итерации, благодаря чему обеспечивается выполнение алгоритмов с хвостовой рекурсией в ограниченном объёме памяти. Такие рекурсивные вычисления, даже если они формально бесконечны (например, когда с помощью рекурсии организуется работа командного интерпретатора, принимающего команды пользователя), никогда не приводят к исчерпанию памяти. Однако, далеко не всегда стандарты языков программирования чётко определяют, каким именно условиям должна удовлетворять рекурсивная функция, чтобы транслятор гарантированно преобразовал её в итерацию. Одно из редких исключений — язык Scheme (диалект языка Lisp), описание которого содержит все необходимые сведения.

Любую рекурсивную функцию можно заменить циклом и стеком.

Назад