Система языков программирования

1. Система языков программирования — это набор специализированных программных продуктов, которые являются инструментальными средствами разработчика. Программные продукты данного класса поддерживают все этапы процесса программирования, отладки и тестирования создаваемых программ.
2. Общесистемные закономерности:
   1. Целостность системы – все языки в системе вносят вклад в реализацию итоговой задачи
   2. Аддитивность – целевая задача системы решается только когда используются все языки системы
   3. Система языков программирования иерархична. Есть уровень который отвечает за взаимодействие с пользователем, уровень взаимодействия между языками и т.д.
   4. Система обладает коммуникативностью, все подсистемы и надсистемы связаны друг с другом
   5. Система является эквифинальной. Можно использовать разные подсистемы, компоненты или элементы для достижения одной цели
   6. Система является историчной, постоянно изменяющейся, каждый год языки программирования дорабатываются, появляются новые
3. Основные определения:
   1. Элементами системы являются компилятор, транслятор, интерпретатор. Это самые базовые части системы языков программирования. Каждый язык имеет свой компилятор, транслятор и интерпретатор.
   2. Компонентами системы являются языки программирования. Это независимые части системы, они обладают некоторыми свойствами системы и подзадачами.
   3. Подсистемами являются стеки разработки, например Python - Django язык программирования – библиотека имеют свою подцель и свои свойства
   4. Надсистемой является компьютер, его свойства и взаимодействие с системой ЯП
   5. Целью системы языков программирования является написание программы в соответствии с требованиями.
   6. Структура – компоненты системы образуют структуру, которая решает организацию системы
   7. Связи в языках программирования
      1. Обмен данными между разными частями программ
      2. Обмен данными между программами
      3. Внутренние связи компилятора и транслятора
      4. Синергия разных языков
      5. Синергия компилятора транслятора и интерпретатора
4. Свойства систем:
   1. Эмерджентность - Некоторые свойства системы могут проявляться только на уровне системы в целом и не могут быть объяснены, анализируя её отдельные компоненты. Так, нельзя просто так объяснить процесс доступа к памяти у языка программирования рассматривая только отдельные его части.
   2. Адаптивность - Системы обладают способностью адаптироваться к изменяющимся условиям внешней среды для поддержания своей целостности и функциональности. Так, меняются предпочтительные языки программирования и версии компиляторов
   3. Интегративность – это её способность объединять или интегрировать различные элементы и компоненты таким образом, чтобы образовывалась функциональная и взаимосвязанная система в целом. Этот аспект является ключевым для обеспечения согласованности и эффективности работы системы. Так, языки программирования можно объединять чтобы достичь поставленную цель
   4. Синергичность системы означает, что взаимодействие между её элементами приводит к созданию дополнительной ценности или эффекта, который не мог бы быть достигнут при работе отдельных компонентов. Так, для написания веб сайта используются различные языки, и невозможно написать сайт на одном только языке.
5. Классификация систем:
   1. По природе системы:
      1. Физические системы: Включают материальные объекты, например, машины, технические устройства и т.д.
      2. Абстрактные системы: Включают концептуальные и информационные системы, например, бизнес-процессы, программные системы и т.д.
   2. По степени сложности:
      1. Простые системы: Состоят из небольшого числа элементов с прямыми взаимосвязями.
      2. Сложные системы: Включают в себя большое количество элементов, подсистем и сложных взаимосвязей.
   3. По предметной области:
      1. Технические системы: Связаны с инженерными и техническими процессами.
      2. Социальные системы: Описывают взаимодействия и отношения в социуме, например, организации, сообщества и т.д.
   4. По функциональности:
      1. Открытые системы: Обмениваются энергией и информацией с окружающей средой.
      2. Закрытые системы: Изолированы от окружающей среды и не обмениваются ею.

Систему языков программирования можно классифицировать как абстрактную, сложную, техническую, открытую.

1. Основные отношения в теории систем:
   1. Отношение включения – это отношение указывает, что одна часть системы является подмножеством другой. Например, подсистема Django – Python является подмножеством системы языков программирования высокого уровня.
   2. Отношение взаимодействия – это отношение описывает взаимодействие между элементами системы. Взаимодействие может быть прямым (напрямую воздействующие друг на друга) или косвенным (воздействие через промежуточные элементы). Например компилятор Python напрямую взаимодействует с исходным кодом на языке
   3. Отношение зависимости – это отношение указывает, что один элемент системы зависит от другого. Изменение в одном элементе может повлиять на другой элемент. Например в задачах машинного обучения при изменении оптимизатора модели, изменяется скорость и точность предсказывания величины.
   4. Отношение обратной связи – это отношение подчеркивает обратные связи между элементами системы. Обратная связь позволяет системе адаптироваться к изменениям внутри и вне неё.
   5. Отношение иерархии – системы часто организованы иерархически, где элементы могут быть структурированы по уровням. Высшие уровни представляют более абстрактные или обобщенные аспекты, а нижние уровни представляют более детализированные элементы. В системе языков программирования есть языки низкого уровня - Assembler и высокого - C++.
   6. Отношение взаимозаменяемости – это указывает, что некоторые элементы системы могут быть взаимозаменяемыми без значительного влияния на функционирование системы. Так, при создании сайтов можно использовать разные языки программирования, но результат не изменится.
   7. Отношение синергии – синергия возникает, когда взаимодействие элементов системы создает дополнительное значение или усилие, которое превосходит сумму их индивидуальных вкладов. Например использование стека C++ и Python при создании игр является более оптимальным выбором чем каждый из этих языков в одиночестве.
   8. Отношение эквивалентности – оно указывает, что некоторые элементы системы могут быть рассмотрены как эквивалентные в определенном контексте или с определенной точки зрения. Например компиляторы разных версий с точки зрения того, какой язык они обрабатывают, считаются эквивалентными.
2. Связь в системном анализе: Понятие "связи" в теории систем относится к взаимодействиям и взаимосвязям между элементами системы. Связи определяют, как элементы системы воздействуют друг на друга, обмениваются информацией и взаимодействуют с окружающей средой. В теории систем связи рассматриваются как ключевой аспект, влияющий на поведение и функционирование системы в целом.
   1. Взаимодействие: Связи определяют степень взаимодействия между элементами системы. Это включает в себя обмен энергией, информацией, воздействие и другие формы взаимодействия. Например связь языков с компиляторами
   2. Обмен информацией: Связи позволяют элементам системы обмениваться информацией. Это может быть критическим для координации действий, принятия решений и обеспечения согласованности в системе. Например, обмен ассемблерных команд с памятью компьютера.
   3. Внутренние и внешние связи: Внутренние связи существуют между элементами внутри системы, в то время как внешние связи определяют взаимодействие системы с окружающей средой. Например связь программы и пользователя как внешняя связь, и связь сервера с клиентским приложением как внутренняя.
   4. Силы и направления связей: В связях могут проявляться различные силы и направления. Например, связь может быть однонаправленной или двунаправленной, сильной или слабой.
   5. Структура связей: Элементы системы могут быть связаны различными способами, образуя структуру связей. Это включает в себя иерархии, сетевые взаимосвязи, кольца обратной связи и другие формы структурирования связей.
   6. Динамические связи: Важным аспектом связей является их динамический характер. Связи могут меняться со временем, что влияет на эволюцию системы. Например переход к другому языку программирования в программе.
   7. Синергия: Связи могут также приводить к синергии, создавая дополнительное значение в результате взаимодействия элементов системы.
3. Структура в системном анализе: В системном анализе понятие "структуры" относится к организации элементов системы и их взаимосвязям. Структура определяет, как элементы системы соединены и как они взаимодействуют друг с другом. Рассмотрение структуры системы позволяет понять её компоненты, подсистемы и их взаимосвязи.
   1. Элементы системы: Структура включает в себя компоненты или элементы системы. Элементы могут быть физическими объектами, процессами, данными, людьми и другими составляющими системы.
   2. Связи между элементами: Структура определяет связи и отношения между элементами системы. Эти связи могут быть прямыми или косвенными, и они формируют сеть взаимодействий в системе. Например в языках программирования компилятор связан с интерпретатором прямой связью, а исходный файл с компилятором косвенно.
   3. Иерархия: Системы часто имеют иерархическую структуру, где элементы группируются по уровням. Высшие уровни представляют более обобщенные аспекты системы, а нижние уровни - более детализированные. Так в языках низкого уровня, например Assembler, доступ к памяти происходит не автоматически, а программист сам задаёт нужные ячейки, а в Python все связи с памятью происходят автоматически.
   4. Подсистемы: Структура может включать в себя подсистемы - части системы, которые могут рассматриваться как отдельные системы собственного рода. Подсистемы могут иметь свою структуру и функциональность. Примеры подсистем:
      1. Стек разработки веб приложений JavaScript, HTML, PHP
      2. Стек разработки игр: Python, C++, Unity, Java
      3. Стек разработки микропроцессоров: C++, Quartus
   5. Функциональные блоки: Элементы системы могут группироваться в функциональные блоки с определенными целями и функциями. Так, например, при разработке приложений сетевую часть часто пишут на языке Python, а логику на C++ или Java
   6. Организационные отношения: В структуре системы учитываются организационные аспекты, такие как структура управления, распределение обязанностей и коммуникационные каналы. Так, управлением языками программирования занимается надсистема центрального процессора и/или компьютер.
   7. Форма и конфигурация: Структура также описывает форму и конфигурацию системы. Это включает в себя расположение элементов, их внутренний порядок и общую организацию.
4. Уровень императивного и декларативного программирования

Императивное и декларативное программирование представляют два основных подхода к написанию программного кода. Они определяют стиль описания задач и выполнения операций в программе.

Императивное программирование – в этом подходе программист явно указывает последовательность шагов, которые компьютер должен выполнить для достижения конечного результата.

Декларативное программирование – в этом подходе программист описывает, что должна делать программа, но не указывает, как это делать шаг за шагом. Вместо этого программист описывает желаемый результат.

Функциональное программирование – это парадигма программирования, в которой программа рассматривается как набор математических функций, которые манипулируют данными. Основные принципы функционального программирования включают в себя:

1. Функции первого класса: Функции в функциональном программировании рассматриваются как "первоклассные объекты", что означает, что они могут быть переданы как аргументы другим функциям, возвращены из функций и присвоены переменным.
2. Неизменяемость: Данные, после того как они были созданы, не могут быть изменены. Вместо этого каждая операция создает новые данные. Это помогает избежать многих ошибок и упрощает анализ программ.
3. Отсутствие побочных эффектов: Функции в функциональном программировании стремятся избегать изменения состояния программы или окружения вне пределов функции. Это уменьшает сложность программы и делает ее более предсказуемой.
4. Рекурсия: Функциональное программирование активно использует рекурсию вместо циклов. Это подчеркивает математический характер функционального программирования и позволяет писать более декларативный и компактный код.
5. Функциональные комбинаторы: Функциональные комбинаторы - это маленькие, часто используемые функции, которые комбинируются в более сложные функции. Это способствует созданию композиции функций и повторному использованию кода.
6. Высшие порядковые функции: Возможность передавать функции как аргументы и возвращать их из других функций. Это поддерживает создание абстракций и повторного использования кода.

Функциональное программирование является подсистемой декларативного программирования.

Процедурное программирование - это структурный подход к написанию программного кода, основанный на использовании процедур (также называемых функциями, подпрограммами или методами). Основные характеристики процедурного программирования включают в себя:

1. Процедуры: Программа разбивается на набор процедур, каждая из которых выполняет конкретную задачу. Процедуры могут принимать аргументы и возвращать значения.
2. Структуры данных: Процедурное программирование обычно использует структуры данных, такие как массивы и записи, для хранения и организации информации.
3. Последовательное выполнение: Инструкции выполняются последовательно от начала до конца программы, за исключением случаев использования условных операторов и циклов.
4. Управление потоком: Программы могут использовать условные операторы (if-else) и циклы (for, while) для управления потоком выполнения в зависимости от определенных условий.
5. Подпрограммы: Модульность достигается с использованием подпрограмм, которые могут быть вызваны из различных частей программы. Это способствует повторному использованию кода.
6. Присваивание: Используется операция присваивания для изменения значений переменных.
7. Процедурные вызовы: Программа выполняется путем последовательного вызова процедур, каждая из которых выполняет конкретную задачу.

Процедурное программирование является подсистемой императивного программирования

Примеры языков программирования:

Декларативное (Функциональное): SQL, Haskell, Skala

Императивное (Процедурное): C++, Java, Python

Схема системы

