**Лекция 1. Парадигмы программирования**

Парадигма программирования - это совокупность идей и понятий, определяющих стиль написания программ. Парадигма в первую очередь определяется базовой программной единицей (определение - декларативное, функциональное программирование, действие - императивное программирование, правило - продукционное программирования, диаграмма переходов - автоматное программирования и др. Сущности). Очень часто парадигма программирования определяется набором инструментов программиста, а именно, на языке программирования и используемыми библиотеками. Многие современные языки программирования является мультипарадигменнимы, то есть допускают использование различных парадигм. Напр., На языке Си, не является объектно-ориентированным, можно писать объектно-ориентированным образом.

**Основные парадигмы программирования:**

**- Императивное (процедурное) программирование.**

**- Структурное программирование.**

**- Функциональное программирование(модульное).**

**- Логическое программирование.**

**- Объектно-ориентированное программирование:**

1) программирование, основанное на классах;

2) программирование, основанное на прототипах;

3) субъектно-ориентированное программирование.

**Императивное программирование** - парадигма программирования, согласно которой описывается процесс получения результатов как последовательность инструкций изменения состояния программы. Подобно тому, как с помощью повелительного наклонения в языкознании перечисляется последовательность действий, необходимо выполнить, императивные программы является последовательностью операций компьютеру для выполнения. Императивные языки основаны на идее переменной, значение которой меняется присвоением. Они называются императивными (лат. Imperative - повелительное), поскольку состоят из последовательностей команд, которые обычно содержат присвоения переменных <назва\_зминнои> = <выражение>, где выражение может ссылаться на значения переменных присвоенных предварительным командами.

**Структурное программирование** - парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. Она предложена в 1970-х годах голландским ученым Дейкстра, была разработана и дополнена Никлаус Вирт.

Согласно этой методологии любая программа - это структура создана на основе трех основных конструкций:

• последовательное выполнение - однократное выполнение операций в том порядке, в котором они записаны в тексте программы.

• разветвления - однократное выполнение одной из двух или нескольких операций в зависимости от выполнения определенной заданной условия.

• цикл - многократное выполнение операции до тех пор пока выполняется заданное условие (условие продолжения цикла)

Каждая конструкция представляет собой блок с одним входом и одним или

несколькими выходами.

**Блок Следование** предполагает линейное выполнения операторов программы.

**Блок Выбор** представляет собой точку принятия решения о дальнейшем ходе выполнения операторов программы.

Выбор осуществляется одной из трех структур:

• if (единственный выбор)

• if ... else (двойной выбор)

• switch или case (множественный выбор)

Все три структуры при желании можно свести к одной типа if.

**Блок Повторение** реализуется одним из трех способов:

• структура while

• структура do / while

• структура for

Все три структуры можно свести к структуре while.

Структурированная программа состоит из вышеназванных блоков по двум правилам: пакетирование (выход одного блока соединяется с входом следующего) и вложения (БАДов конструкции могут быть вложены друг в друга произвольным образом).

Таким образом, структурированные программы содержат всего семь типов управляющих структур, которые соединяются всего двумя способами.Такие программы легко создаются и тестируются. Разработка занимает меньше времени. Программы более прозрачные и легко поддаются переработке. Повторяющиеся фрагменты программы (или не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться в виде подпрограмм (процедур или функций). В этом случае в тексте основной программы, вместо помещенного в подпрограмму фрагмента, вставляется инструкция вызова подпрограммы. При выполнении такой инструкции выполняется вызвана подпрограмма, после чего выполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы.

**Теорема о структурном программировании:**

Эта теорема была сформулирована итальянскими математиками К.Бомом и Дж.Якобини в 1966 году и говорит нам о том, как можно избежать использования оператора перехода goto.

Любую схему алгоритма можно представить в виде композиции вложенных блоков begin и end, условных операторов if, then, else, циклов с предусловием (while) и возможно дополнительных логических переменных (флагов).

Перечислим некоторые достоинства структурного программирования:

1.Структурне программирования позволяет значительно сократить число вариантов построения программы по одной и той же спецификации, что значительно снижает сложность программы и, что еще важнее, облегчает понимание ее другими разработчиками.

2. В структурированных программах логически связанные операторы находятся визуально ближе, а слабо связаны - дальше, что позволяет обходиться без блок-схем и других графических форм изображения алгоритмов (по сути, сама программа является собственной блок-схеме).

3.Сильно упрощается процесс тестирования и отладки структурированных программ.

**Функциональное программирование** - парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании). Противопоставляется парадигме императивного программирования, которая описывает процесс вычислений как последовательность смены состояний (в смысле, подобном таком же в теории автоматов).

Функциональная программа представляет собой определение функций. Функции определяются через другие функции или рекурсивно - через себя. В процессе выполнения программы, функции получают параметры, вычисляют и возвращают результат, в случае необходимости вычисляя значение других функций. Программируя на функциональном языке, программист не должен описывать порядок вычислений. Ему необходимо просто описать желаемый результат в виде системы функций. Почти каждый из нас так или иначе использовал функциональный подход к программированию. Возьмем к примеру, всем известный Microsoft Excel. Записывая содержание ячейки в виде, похожем на обычную математическую формулу, мы не задумываемся о действительном порядка вычислений этой формулы, возлагая эти функции на встроенный в Excel интерпретатор. Другим простым примером могут послужить языка запросов к базам данных. Языка основаны на реляционном исчислении (например, всем известный SQL или QBE) описывают только результат, являясь тем самым по сути функциональными.

Программы на функциональных языках обычно намного короче и проще, чем те же программы на императивных языках.

Кроме упрощения анализа программ еще одно весомое преимущество - параллелизм. Раз все функции для вычислений используют только свои параметры, мы можем вычислять независимые функции в произвольном порядке или, скажем, параллельно, на результат вычислений это не повлияет. Причем параллелизм этот может быть организован не только на уровне компилятора с языка, но и на уровне архитектуры.

Наиболее известными языками функционального программирования является языка:

• LISP - считается первой функциональной языке программирования.

• Haskell - функциональный язык программирования. Названа в честь Хаскелл Карри. Последний стандарт языка - Haskell 98.

• Erlang - (Joe Armstrong, 1986) функциональный язык с поддержкой процессов.

• F # - функциональный язык для платформы .NET.

ФП в нефункциональных языках

В языке C указатели на функцию могут быть использованы для получения эффекта функций высшего порядка. Функции высшего порядка и отложена списочная структура реализованы в библиотеках С ++. В языке C # версии 3.0 и выше можно использовать λ-функции для написания программы в функциональном стиле.

**Логическое программирование** - парадигма программирования, основанная на выводе из начальных фактов новых фактов по заданным логическим правилам.

Логическое программирование основано на теории математической логики. Самой известной языком логического программирования является Prolog, что является по своей сути универсальной машиной выводу, что работает в предположении замкнутости системы фактов.

**Объектно-ориентированное программирование (ООП)** - одна из парадигм программирования, которая рассматривает программу как множество «объектов», взаимодействующих между собой. Сегодня многие языки программирования (в частности, Java, C #, C ++, Python, PHP, Ruby и Objective-C, ActionScript 3) поддерживают ООП.

В отличие от традиционных взглядов, когда программу рассматривали как набор подпрограмм, или как перечень инструкций компьютеру, ООП программы можно считать совокупностью объектов. Согласно парадигмы объектно-ориентированного программирования, каждый объект способен получать сообщения, обрабатывать данные, и отправлять сообщения другим объектам. Каждый объект - своеобразный независимый автомат с отдельным назначением и ответственностью.

Класс. Класс определяет абстрактные характеристики некоторой сущности, включая характеристики самой сущности (ее атрибуты или свойства) и действия, которые она способна выполнять (его поведение, методы или возможности). Например, класс Собака может характеризоваться чертами, присущими всем собакам, в частности: порода, цвет меха, способность лаять. Классы вносят модульность и структурированность в объектно-ориентированную программу. Как правило, класс должен быть понятным для программистов, которые знают предметную область, что, в свою очередь, значить, что класс должен иметь значение в контексте. Также, код реализации класса должна быть достаточно самодостаточным. Свойства и методы класса, вместе называются его членами.

**Объект**. Отдельный **экземпляр** класса. Класс Собака отвечает всем собакам путем описания их общих черт; объект Сирко является одним отдельным собакой, отдельным вариантом значений характеристик. Собака имеет мех; Серо имеет коричнево-белый мех. Объект Сирко является экземпляром (экземпляром) класса Собака. Совокупность значений атрибутов отдельного объекта называется состоянием.

**Метод.** Возможности объекта. Поскольку Сирко - Собака, он может лаять. Поэтому лаять () является одним из методов объекта Сирко. Он может иметь и другие методы, в частности: место () или есть (). В рамках программы, использование метода должен влиять только один объект; все Собаки могут лаять, али надо чтобы лаяла только одна отдельная собака.