Парадигмы программирования

Своим современным значением в научно-технической области термин «парадигма» обязан, по-видимому, Томасу Куну и его книге «Структура научных революций». Кун называл парадигмами устоявшиеся системы научных взглядов, в рамках которых ведутся исследования. Согласно Куну, в процессе развития научной дисциплины может произойти замена одной парадигмы на другую (как, например, геоцентрическая небесная механика Птолемея сменилась гелиоцентрической системой Коперника), при этом старая парадигма ещё продолжает некоторое время существовать и даже развиваться благодаря тому, что многие её сторонники оказываются по тем или иным причинам неспособны перестроиться для работы в другой парадигме.

Парадигма програмирования — совокупность базовых идей и понятий, определяющая стиль написания программ. Часто путают с методологией, которая определяет способы решения конкретных инженерных проблем. Вообще говоря, разделение на парадигмы весьма условно. Тем не менее, можно выделить несколько основных. Собственно, к программированию существует два основных подхода: императивный и декларативный. При императивном программировании программа представляется последовательностью операторов, меняющих состояние вычислителя (то есть, программа — последовательность команд исполнителю). Декларативное программирование предполагает описание того, что должна делать программа, опуская подробности того, как она должна это делать.

Существует несколько математических моделей понятия "вычисление". Все они эквивалентны, но описывают вычисления с разных точек зрения. Из различных математических моделей следуют различные парадигмы. Так, для императивного программирования математической моделью является машина Тьюринга (точнее, её вариация — машина Поста). Машина Тьюринга была предложена в 1937 году Аланом Тьюрингом для формализации понятия "алгоритм". Устроена она следующим образом.

- В состав машины Тьюринга входит бесконечная в обе стороны лента (возможны машины Тьюринга, которые имеют несколько бесконечных лент), разделённая на ячейки, и управляющее устройство, способное находиться в одном из множества состояний. Число возможных состояний управляющего устройства конечно и точно задано.
- Управляющее устройство может перемещаться влево и вправо по ленте, читать и записывать в ячейки ленты символы некоторого конечного алфавита. Выделяется особый пустой символ, заполняющий все клетки ленты, кроме тех из них (конечного числа), на которых записаны входные данные.
- Управляющее устройство работает согласно правилам перехода, которые представляют алгоритм, реализуемый данной машиной Тьюринга. Каждое правило перехода предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей клетке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево или вправо. Некоторые состояния машины Тьюринга могут быть помечены как терминальные, и переход в любое из них означает конец работы, остановку алгоритма.

Машина Тьюринга называется детерминированной, если каждой комбинации состояния и ленточного символа в таблице соответствует не более одного правила. Если существует пара (ленточный символ -- состояние), для которой существует 2 и более команд, такая машина Тьюринга называется недетерминированной.

Нужно это для доказательства различных утверждений об алгоритмах. Полагается за аксиому, что любой алгоритм может быть представлен в виде машины Тьюринга. Так что если построить для

данного алгоритма реализующую его машину Тьюринга, можно, например, формально показать, что он завершится, исследовать время его работы и занимаемую память. Ещё полезное свойство — Тьюринг-полнота — вычислительная система называется полной по Тьюрингу, если на ней можно вычислить всё, что можно вычислить на машине Тьюринга, то есть всё вообще. Простым способом доказательства тьюринг-полноты является эмуляция машины Тьюринга на проверяемой вычислительной системе.

У машины Поста есть программа в явном виде, и в явном виде данные, так что она больше похожа на "настоящий" компьютер. На основе математической модели Поста была предложена так называемая архитектура фон Неймана (1946 год):

- Принцип программного управления. Программа состоит из набора команд, которые выполняются процессором друг за другом в определенной последовательности.
- Принцип однородности памяти. Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления — чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными.
- Принцип адресуемости памяти. Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.
- Принцип последовательного программного управления. Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.

По такому принципу устроено подавляющее большинство современных компьютеров, хотя она ещё с 60-х годов подвергается критике. Понятно поэтому, почему императивный подход к программированию столь популярен.

Структурное программирование

"Чистое" императивное программирование не может рассматриваться как парадигма, поскольку не предполагает особого "стиля". Долгое время, пока возможности вычислительных систем были ограничены и программы были небольшие, программировали "как придётся". Первый язык программирования высокого уровня (фортран) появился только в 50-х годах (54-57 год, Дж. Бэкус). Однако, писать программы "полотном" оказалось не очень удобно (так называемый спагетти-код), поэтому начали пытаться как-то упорядочить эту деятельность. Так зародилась парадигма структурного программирования (это чем мы занимались весь семестр). Структурное программирование пришло на смену неструктурированному программированию в начале 70-х. Любая программа может быть представлена как комбинация последовательно исполняемых операторов, ветвлений, итераций. См. статью Дейкстры «Go To Statement Considered Harmful» (1968г). Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться в виде т. н. подпрограмм (процедур или функций). В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция вызова подпрограммы. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы. Разработка программы ведётся пошагово, методом «сверху вниз». Языки-представители: Алгол, Паскаль, C, Модула-2, Ада.

Объектно-ориентированное программирование

Первый ОО-язык – Симула-67, были и более ранние разработки. Популярной парадигма стала только в середине 90-х, развитие связано с широким распространением графических интерфейсов и компьютерных игр. Программа представляет собой набор объектов, объекты взаимодействуют путём

посылки сообщений по строго определённым интерфейсам, имеют своё состояние и поведение, каждый объект является экземпляром некоего класса (это опционально, бывают языки с прототипами вместо классов).

Основные принципы ООП

1. Инкапсуляция — тут ей обычно называют сокрытие реализации от пользователя. Пользователь может взаимодействовать с объектом только через интерфейс. Позволяет менять реализацию объекта, не модифицируя код, который этот объект использует.
2. Наследование позволяет описать новый класс на основе существующего, наследуя его свойства и функциональность. Наследование — отношение «является» между классами, с классом-наследником можно обращаться так же, как с классом-предком.
3. Полиморфизм — классы-потомки могут изменять реализацию методов класса-предка, сохраняя их сигнатуру. Клиенты могут работать с объектами класса-родителя, но вызываться будут методы класса-потомка (позднее связывание).

Например:

```
class Animal
{
	public:
	Animal(const string& name) : name(name) {}
	virtual string talk() = 0;
	 void rename(string newName);
	const string name;
};
class Cat : public Animal
{
	public:
	Cat(const string& name) : Animal(name) {}
	string talk() { return "Meow!"; }
};
class Dog : public Animal
{
	public:
	Dog(const string& name) : Animal(name) {}
	string talk() { return "Arf! Arf!"; }
};
```

[Тут следует рассказ о конструкторах, деструкторах, виртуальных методах, позднем и раннем связывании]

Ещё так можно сделать вычисление дерева арифметического выражения. Каждый узел дерева представляется объектом с методом calculate, операнды возвращают своё значение, операции - результат применения. Тогда для вычисления не надо писать никаких свичей, достаточно просто вызвать метод calculate у корня.

Языки-представители: Java, C++, C#, Object Pascal / Delphi language, Smalltalk.

Функциональное программирование

Вычисления рассматриваются как вычисления значения функций в математическом понимании (без побочных эффектов). Основано на λ-исчислении Чёрча — ещё одной формализации понятия "вычисление". λ-исчисление основано на функциях.

Пример: λx.2*x+1 — функция от аргумента x, вычисляющая 2*x+1.

Функции могут принимать функции в качестве параметров и возвращать функции в качестве результата. Функция от n переменных может быть представлена, как функция от одной переменной, возвращающая функцию от n-1 переменной (карринг). Тут довольно много довольно хорошо проработанной математической теории. В основе лямбда-исчисления лежат две базовые операции — аппликация и абстракция. Аппликация — вызов функции с заданным значением аргумента. Её обычно обозначают f a, где f — функция, a — аргумент. f трактуется как алгоритм, вычисляющий результат по данному входному значению. Абстракция (или лямбда-абстракция) строит функции по заданным выражениям. Если t[x] — выражение, "свободно" содержащее x, то λx.t[x] обозначает функцию x -> t[x].

β-редукция: поскольку выражение λx.2*x+1 обозначает функцию, ставящую в соответствие каждому x значение 2*x+1, то для вычисления выражения, в которое входят и аппликация и абстракция, необходимо выполнить подстановку числа 3 в терм 2*x+1 вместо переменной x. В результате получается 2*3+1 = 7. Это соображение в общем виде записывается как (λx.t) a = t[x:=a] и носит название β-редукция. Выражение вида (λx.t) a, то есть применение абстракции к некому терму, называется редексом (redex). Несмотря на то, что β-редукция по сути является единственной «существенной» аксиомой λ-исчисления, она приводит к весьма содержательной и сложной теории. Вместе с ней λ-исчисление обладает свойством полноты по Тьюрингу и, следовательно, представляет собой простейший язык программирования.

На этой теории довольно просто построить языки программирования, первый из которых (LISP) был разработан в 1958 году и активно используется по сей день. Эти языки уже совсем не императивные, поскольку понятием "состояние" не обладают. Оператора присваивания там тоже нет, как и типичных для императивных языков конструкций, типа циклов. Порядок вычислений тоже не определён.

Особенности:

- Программы не имеют состояния и не имеют побочных эффектов
- Порядок вычислений не важен
- Циклы выражаются через рекурсию
- «Ленивые» вычисления
- Формальные преобразования программ по математическим законам

Примеры программ (на языке Haskell)

Факториал:

```
fact :: Integer -> Integer
fact 0 = 1
fact n | n > 0 = n * fact (n - 1)
```

QSort:

```
sort [] = []
sort (pivot:rest) = sort [y | y <- rest, y < pivot]
```

```
                    ++ [pivot]
                    ++ sort [y | y <- rest, y >=pivot]
```

Получение числа четных чисел в списке:

```
numberOfEven = length . filter even
```

Языки-представители: Лисп (LIst PRocessing), ML (OCaml), Haskell, Erlang, F#

Логическое программирование

Парадигма программирования, основанная на автоматическом доказательстве теорем. Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций. Программа представляет собой набор фактов и правил, система сама строит решение с использованием правил логики. Создавалось в 60-х для решения задач искусственного интеллекта и экспертных систем. Самым известным логическим языком программирования считается Пролог (начало 70-х), основан на математической модели дизъюнктов Хорна (вариант исчисления предикатов).

Основными понятиями в языке Пролог являются факты, правила логического вывода и запросы, позволяющие описывать базы знаний, процедуры логического вывода и принятия решений.

Факты в языке Пролог описываются логическими предикатами с конкретными значениями. Правила в Прологе записываются в форме правил логического вывода с логическими заключениями и списком логических условий.

Факты в базах знаний на языке Пролог представляют конкретные сведения (знания). Обобщённые сведения и знания в языке Пролог задаются правилами логического вывода (определениями) и наборами таких правил вывода (определений) над конкретными фактами и обобщенными сведениями.

Пример программы на Прологе:

```
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).
parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).
```

Пример запроса:

```
?- sibling(sally, erica).
Yes

?- father_child(Father, Child).
```

Быстрая сортировка на Прологе:

```
partition([], _, [], []).
partition([X|Xs], Pivot, Smalls, Bigs) :-
```

```
    (   X @< Pivot ->
        Smalls = [X|Rest],
        partition(Xs, Pivot, Rest, Bigs)
    ;   Bigs = [X|Rest],
        partition(Xs, Pivot, Smalls, Rest)
    ).

quicksort([])     --> [].
quicksort([X|Xs]) -->
    { partition(Xs, X, Smaller, Bigger) },
    quicksort(Smaller), [X], quicksort(Bigger).
```

Стековое программирование

Форт — разработан в 60-х Чарльзом Муром «для себя». Широко распространён для программирования встроенных систем и задач, естественным образом выражающихся в терминах стеков. Основной элемент программы: слово. Форт-система состоит из словаря (набора слов) и стеков — арифметического и командного (с их помощью производятся вычисления). Используется обратная польская нотация. Примеры:

```
25 10 * 50 + .
```

Вывод:

```
300 ok
```

```
: FLOOR5 ( n -- n' )   DUP 6 < IF DROP 5
    ELSE 1 - THEN ;
```

- то же самое на C++:

```
int floor5(int v) { return v < 6 ? 5 : v - 1; }
```

- более красиво на Форте:

```
: FLOOR5 ( n -- n' ) 1- 5 MAX ;
: HELLO  ( -- )  CR ." Hello, world!" ;
```

Автоматное программирование

Автоматное программиирование — это парадигма программирования, при использовании которой программа или её фрагмент осмысливается как модель какого-либо формального автомата.

В зависимости от конкретной задачи в автоматном программировании могут использоваться как конечные автоматы, так и автоматы более сложной структуры.

Определяющими для автоматного программирования являются следующие особенности:

- временной период выполнения программы разбивается на шаги автомата, каждый из которых представляет собой выполнение определённой (одной и той же для каждого шага) секции кода с единственной точкой входа; такая секция может быть оформлена, например, в виде отдельной функции и может быть разделена на подсекции, соответствующие отдельным состояниям или категориям состояний
- передача информации между шагами автомата осуществляется только через явно обозначенное множество переменных, называемых состоянием автомата; между шагами

автомата программа (или её часть, оформленная в автоматном стиле) не может содержать неявных элементов состояния, таких как значения локальных переменных в стеке, адреса возврата из функций, значение текущего счётчика команд и т.п.; иначе говоря, состояние программы на любые два момента входа в шаг автомата могут различаться между собой только значениями переменных, составляющих состояние автомата (причём такие переменые должны быть явно обозначены в качестве таковых).

Полностью выполнение кода в автоматном стиле представляет собой цикл (возможно, неявный) шагов автомата.