Вводная лекция по функциональному программированию

Модели вычислений и парадигмы программирования

Основные стили, или парадигмы программирования, к которым обычно относят императивное, функциональное, логическое и объектно-ориентированное программирование, возникли более сорока лет назад вместе с первыми языками программирования и развивались сначала относительно независимо друг от друга. Каждая из парадигм отображает определенную модель вычислений, включая структуры данных и механизмы управления, и с ней связан определенный класс прикладных задач, которые удобно решать средствами данной парадигмы.

В настоящий момент большинство современных языков программирования обычно включают средства и приёмы программирования различных парадигм, хотя классифицируются согласно средствам своего ядра (к примеру, язык программирования Лисп – функциональный язык, хотя включает некоторые конструкции императивного стиля).

Императивная, или процедурная парадигма программирования, представляемая такими языками как Фортран, Алгол, Паскаль, является наиболее распространенной. Она развилась на базе низкоуровневых языков (машинные коды, ассемблер), основанных на архитектуре фон Неймана. Императивная программа состоит из последовательно выполняемых команд и вызовов процедур, которые производят обработку данных и изменение значений переменных программы. Переменные при этом рассматриваются как некоторые контейнеры для данных, подобно ячейкам памяти компьютера.

Функциональная парадигма программирования является существенно менее традиционной, и в тоже время более древней, поскольку получила развитие из вычислений по алгебраическим формулам. Функциональная программа состоит из набора взаимосвязанных и, как правило, рекурсивных функций. Каждая функция определяется выражением, которое задает правило вычисления её значения в зависимости от значений ее аргументов и которое является композицией встроенных функций, а также других функций, описанных в программе. Выполнение функциональной программы заключается в последовательном вычислении значений функциональных вызовов.

В еще менее традиционной и необычной логической парадигме программа рассматривается как множество логических формул: аксиом (фактов и правил), описывающих свойства некоторых объектов, и теоремы, которую необходимо доказать. В свою очередь, выполнение программы – это доказательство теоремы, в ходе которого строится объект с описанными свойствами.

Основные различия указанных парадигм программирования касаются не только концепции программы и принципов её выполнения, но и роли переменной в ходе вычислений. В отличие от императивных программ в чисто функциональных и логических программах отсутствует явное присваивание значений переменным и, как следствие, побочные эффекты. Переменные в таких программах подобны переменным в математике: они являются обозначением функциональных аргументов или объктов, конструируемых в процессе доказательства. Еще одна

яркая особенность функциональной и логической парадигм – использование рекурсии вместо циклов. Все эти особенности объясняют, почему эти две парадигмы программирования считают нетрадиционными.

В получающей все большее распространение объектно-ориентированной парадигме программа описывает структуру и поведение вычисляемых объектов и классов объектов. Объект обычно включает некоторые данные (состояние объекта) и операции с этими данными (методы), описывающие поведение объекта. Классы представляют множество объектов со схожей структурой и схожим поведением. Обычно описание классов имеет иерархическую структуру, включающую полиморфизм операций. Выполнение объектно-ориентированной программы представляет собой обмен сообщениями между объектами, в результате которого они меняют свои состояния.

Характерные свойства основных парадигм программирования представлены в Таблице 1.

Таблица 1.

Свойства парадигм программирования

Парадигма	**Ключевой концепт**	**Программа**	**Выполнение программы**	**Результат**
Императивная	Команда (инструкция)	Последова-тельность команд	Исполнение команд	Заключительное состояние памяти (переменных)
Функциональная	Функция	Совокупность функций	Вычисление функций	Значение главной функции
Логическая	Предикат	Логические формулы	Логическое доказатель-ство	Успешность или неуспешность доказательства
Объектно-ориентированная	Объект	Совокупность классов объектов	Обмен сообщениям и между объектами	Результирующее состояние объектов

Теоретические модели вычислений, повлекшие возникновение различных парадигм программирования:

- Машина Тьюринга => императивная парадигма программирования.
- Нормальные алгоритмы Маркова (НАМ) => язык Рефал (функциональная парадигма).
- λ-исчисление Черча-Клини => функциональная парадигма.
- Логика предикатов первого порядка => логическая парадигма программирования.

Тезис Черча-Клини: класс алгоритмически разрешимых арифметических функций совпадает с классом частично рекурсивных арифметических функций.

Есть базовый набор примитивных арифметических числовых функций и операций над функциями – суперпозиция и рекурсия (минимальный набор функций и примитивная рекурсия). Любая частично рекурсивная функция может быть представлена в таком виде.

Особенности функциональной парадигмы программирования

Зачатки функционального стиля программирования есть практически во всех императивных языках программирования. Например, правая часть оператора присваивания

```
X= 2+Y*3
```

по сути является функцией, вычисляющей некоторое значение. Это запись функции в инфиксной форме. В функциональной парадигме принята запись в префиксной форме. Эквивалентной данной выше инфиксной форме будет следующая префиксная запись: `+(2,*(Y,3))`.

Свойство референциальной прозрачности (функциональности): значение выражения однозначно зависит от входящих в него составных частей. Или значение функции однозначно зависит от заданных аргументов (такая функция является чистой).

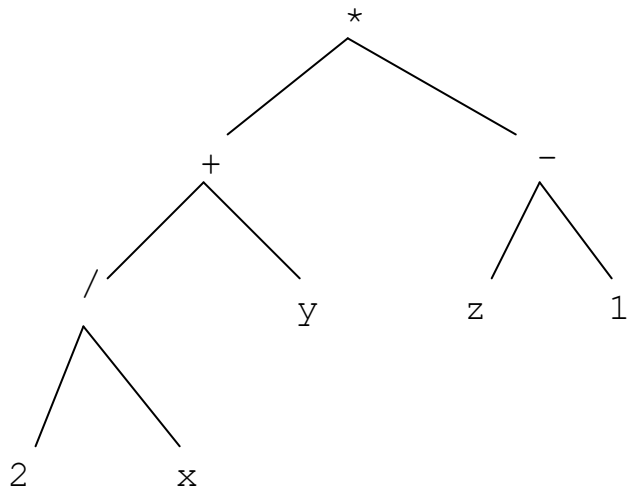
Пример паскалевской функции, не являющейся чистой:

```
function f(n: integer): integer;
  begin if B then f:=2*n else f:=n*n;
        B:= not B
  end;
…
B:=false;
X:=f(2)+f(3)+f(1);
Y:=f(2)+f(3)+f(1);
…
```

Значения переменных `X` и `Y` будут разными, хотя заданы одинаковыми формулами. Проблема – в использовании глобальной переменной `B`, которая влияет на вычисление значения функции `f` и изменяет свое значение после каждого вычисления функции.

В каком-то смысле знакомимся с функциональным стилем еще в школе, изучая алгебраические функции и выражения и вычисляя по формулам.

Например, формула (2/x+y)*(z-1) определяет функцию, здесь используется инфиксная запись арифметических функций. Порядок вычислений по этой формуле удобно представить графически – в виде дерева (четкая древесная структура вычислений):



Такое дерево определяется приоритетом арифметических операций и расставленными скобками.

Другие полезные особенности вычисления по формулам:

- Аппликативная структура формул: формула (выражение) разбивается на составляющие его части, каждая из которых является либо операцией, либо операндом, причем операнд обозначает некоторое значение, а операция – функцию. Следовательно, ничего лишнего! Операнды находятся рядом с операциями, их не надо доставать из общей памяти, результаты операций никуда не надо посылать (они нужны там, где получены).
- Результат вычислений не зависит от допустимого изменения порядка действий (при сохранении иерархии в соответствии с деревом выражения) => отсюда путь к параллельным вычислениям.
- При вычислении значений используется естественный принцип синхронизации вычислений: всякая операция должна ждать завершения вычисления своих операндов – и только. Следствие последних двух пунктов: возможность распараллеливания вычислений (параллельное вычисление операндов).
- Возможность эквивалентных алгебраических преобразований формул (упрощение, раскрытие скобок, приведение подобных) – использование алгебры формул, позволяющей оптимизировать вычисления, устанавлтвать эквивалентность формул, доказывать математические свойства формул и т.п.

Как определить функциональный стиль программирования?

Функциональное программирование – это способ составления программ, в которых единственным

- действием является применение (вызов) функции;

- способом расчленения программы на части является введение имени для функции и задание для этого имени выражения, вычисляющего значение функции;
- правилом композиции является операция суперпозиции функций.

Особенности функциональной парадигмы программирования:
- Функциональность (референциальная прозрачность).
- Отсутствие побочных эффектов.
- Нет оператора присваивания. Присваивание неявное (передача значений через параметры функции).
- Роль переменной – как в математике (для обозначения аргументов функции).
- Данные в ходе вычислений не изменяются, строятся лишь новые структуры => необходимость сборки мусора.

В функциональном программировании вместо цикла используется рекурсия, а вместо условного оператора – условное выражение.

Достоинства функционального программирования:
- декларативность;
- надежность;
- удобство тестирования;
- возможности оптимизации;
- распараллеливание.

Основной недостаток функционального программирования – неэффективность вычислений на машинах фон неймановской архитектуры.

В курсе функционального программирования будут рассмотрены следующие средства программирования:

1. сложные структуры (символьные вычисления) ┐
2. рекурсия ├ Лисп (динамическая типизация)
3. функционалы ┘
4. ленивые вычисления ┐
5. бесконечные списки ├ Хаскель (статическая типизация)
6. автоматический вывод типов ┘

Почему был выбран язык программирования Лисп:
1. Самый старый, исторически первый (60-е годы), давший толчок развитию парадигмы.
2. Обработка символьной информации (сложные символьные структуры).
3. Основной язык программирования задач ИИ до 90-х годов.
4. Оригинальность концепции (единая синтаксическая форма представления данных и программы).
5. Солидный теоретический базис.

6. Естественная расширяемость. Существует целая Лисп-культура: Лисп породил семейство функциональных языков:
   - AutoCAD система автоматизации инженерных расчетов;
   - Emacs текстовый редактор UNIX/Linux.
7. Развитие технологии программирования (системные решения):
   - ссылочная организация памяти;
   - сборка мусора (автоматизация повторного использования памяти);
   - смешанная схема трансляции;
   - макросы;
   - интерпретируемые языки.