Введение в декларативное и функциональное программирование

Какие бывают виды программирования??? :)

Виды программирования

1. wiki: Декларативное программирование
2. wiki: Императивное программирование
3. wiki: Структурное программирование, О структурном программировании
4. wiki: Языково-ориентированное программирование
5. wiki: Логическое программирование, habr: “Что такое логическое программирование и зачем оно нам нужно”
6. wiki: Объектно-ориентированное программирование
7. wiki: Процедурное программирование
8. wiki: Автоматное программирование, Генеративное программирование, Блочное программирование
9. wiki: Генетическое, Метагенетическое программирование (там же) и Генетическое программирование («Yet Another Велосипед» Edition), Эволюционное программирование
10. wiki: Метапрограммирование
11. wiki: Обобщённое программирование
12. wiki: Событийно-ориентированноепрограммирование
13. wiki: Реактивное программирование
14. Параллельное программирование
15. Конкурентное программирование,
    Основные принципы программирования: конкурентность, Parallelism vs Concurrency: правильно подбираем инструменты
16. Асинхронное программирование: концепция, реализация, примеры, Понимание асинхронного программирования, Основные понятия асинхронного программирования, Асинхронное программирование: концепция Deferred
17. Математическое программирование: Выпуклое программирование, Линейное программирование, Дискретное программирование, Параметрическое программирование, Стохастическое программирование, Дробно-линейное программирование, Целочисленное программирование, Нелинейное программирование, Квадратичное программирование, Кубическое программирование, Кубическое программирование. Пост 1, постановка задачи, Динамическое программирование
18. wiki: Квантовое программирование
19. wiki: Нейролингвистическое программирование
20. wiki: Экстремальное программирование

Что такое императивное программирование?

Обычно императивное программирование — это инструкции, описывающее на том или ином уровне детализации, ***как*** решить задачу и представить результат.

ru.wikipedia: Императивное программирование

Основные черты императивных языков:

- использование именованных переменных;
- использование оператора присваивания;
- использование составных выражений;
- использование подпрограмм.

более подробно:

- в исходном коде программы записываются инструкции (команды);
- инструкции должны выполняться (как правило) последовательно;
- при выполнении инструкции данные, полученные при выполнении предыдущих инструкций, могут читаться из памяти;
- данные, полученные при выполнении инструкции, могут записываться в память.

Императивная программа похожа на приказы (англ. imperative — приказ, повелительное наклонение), выражаемые повелительным наклонением в естественных языках, т.е. представляют собой последовательность команд, которые должен выполнить компьютер.

Структура архитектуры фон Неймана

В простейшей форме характеризуется тремя составными частями: центральным процессором, памятью и соединительной шиной, которая за 1 шаг может передавать только одно слово и посылать некий адрес в память… Задача программы состоит в том, чтобы как-то изменить содержимое памяти.

Отвлечемся пока от критики архитектуры, на которой основаны практически все существующие вычислительные устройства.

Традиционные языки программирования

- машинный код;
- ассемблер;
- Fortran, Algol, Basic;
- Pascal, C, Perl;
- Python, Java, C++.

Обычно, современные языки являются высокоуровневыми. Но даже наличие разнообразных нововведений вроде ООП мало меняет ситуацию — они по сути сложные версии машины фон Неймана, т.е. *императивные* или как ещё говорят, *процедурные*.

Вывод

Несмотря на все свою интеллектуальную привлекательность программирование является трудным занятием. После сорока лет многие перестают программировать под предлогом, что устали. Основной причиной такого положения является субъективное чувство

«нарушения энергетического баланса»: много тратят, но мало получают. Программисты тратят много лет и усилий, чтобы научиться программировать в императивном (процедурном) стиле. Но императивный стиль программирования — стиль, в котором выполняются программы на компьютере — совершенно несвойственен человеческой природе. Традиционное процедурное программирование приучает программиста мыслить понятиями «машины фон Неймана», а не математическими понятиями. Традиционный программист по своим знаниям, в первую очередь, должен быть инженером.

Процедурное программирование наиболее пригодно для решения задач, в которых последовательное исполнение каких-либо команд является естественным. Примером здесь может служить управление современными аппаратными средствами. Поскольку практически все современные компьютеры императивны, эта методология позволяет порождать достаточно эффективный исполняемый код. С ростом сложности задачи процедурные программы становятся все менее и менее читаемыми. Программирование и отладка действительно больших программ (например, компиляторов), написанных исключительно на основе методологии императивного программирования, может затянуться на долгие годы.

> «Я с лёгкостью сделаю на C++ за месяц то, что вы с трудом напишите на Lisp за неделю…»

ВОЗ считает, что выгорание на работе стало большой проблемой

Декларативное программирование

ru.wikipedia: Декларативное программирование

Парадигма программирования, в которой задаётся спецификация решения задачи, т.е. описывается, **что** представляет собой проблема и ожидаемый результат.

Декларативное программирование стремится описывать саму задачу в терминах, во-первых, самой задачи, а во-вторых, математики, логики и, возможно когда-либо, на естественном языке (а в настоящее время на вариантах DSL).

ru.wikipedia: Языково-ориентированное_программирование

В идеале, составляется точная спецификация задачи, а машина затем «думает» как решать задачу и сама её решает.

На практике же мы можем указать два базовых направления декларативного программирования: логическое и функциональное.

Логическое программирование

Парадигма программирования, основанная на автоматическом доказательстве теорем. Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций. Самым известным языком логического программирования является Prolog.

…

Используется язык исчисления логики предикатов первого порядка. Предполагалось, что программист не указывает последовательность шагов решения, а на логическом языке

описывал бы свойства интересующей его области. Далее, исполняющая система пыталась бы найти решение сама.

Несмотря на использование ряда интересных идей (и на взрыв интереса к языку Prolog в конце 80-х гг. прошлого века из-за объявления японцами Prolog'а языком для своих (супер?)компьютеров 5-го поколения), по сути вычисления на языке Prolog остались переборными и в большинстве практических задач малоэффективными.

Факт

Развитие логического и семантического программирования в новосибирском академгородке с начала 80-х гг. и до сих пор.

SQL

Ещё одним примером декларативного программирования является «информационно-логический язык» (в другом представлении — DSL) SQL, предназначенный для работы с данными, хранимыми в реляционных базах данных. Этот язык не является тьюринг-полным, но это декларативный язык программирования, не детализирующий как именно следует выполнять запросы, что ложится уже на плечи СУБД.

en.wikipedia: SQL

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).

Разница в понимании функций в функциональном и в императивном программировании

Рассмотрим следующую программу на языке JavaScript:

```
function f(n) {
  if (flag) {
      flag = !flag; return(n)
  } else {
      flag = !flag; return(2*n)
  }
}

var flag = true;

console.log( f(1)+f(2) );
console.log( f(2)+f(1) );
```

при исполнении в консоли увидим следующие ответы:

```
5
4
```

хотя, казалось бы, на первый взгляд, мы могли бы ожидать коммутативности сложения! Разумеется, дело не в изменении свойств операции «+» в JavaScript, просто, так называемая функция `f` в своём коде помимо аргумента использовала значение глобальной переменной `flag`. Такие функции мы будем назвать «нечистыми», или *функциями с побочными эффектами*.

Отметим, что в данном случае, ещё одним источником проблемы стал оператор присваивания «=» в JavaScript (или «:=» в Pascal).

Аналог примера на Pascal:

```
program ex1(output);
var flag: boolean;

function f(n: integer): integer;
begin
  if flag then f:=n
          else f:=2*n;
  flag := not flag;
end;

begin
  flag := true;
  writeln( f(1)+f(2) );
  writeln( f(2)+f(1) );
end.
```

В целом, такие ситуации называются *побочными эффектами*.

ru.wikipedia: Побочный эффект

Точнее, побочными эффектами обычно называют возможность функций в процессе выполнения своих вычислений читать и модифицировать значения глобальных переменных, осуществлять операции ввода-вывода, реагировать на исключительные ситуации и т.п. При вторичном вызове подобной функции (с одними и теми же входными аргументами) мы можем обнаружить, что в результате будут возвращены разные значения.

Побочные эффекты не являются чем-то слишком необычным. В императивных языках они встречаются чаще (можно сказать повсеместно), в декларативных и функциональных языках — минимальным образом. Но они все-таки необходимы и в последних. В первую очередь это относится именно к вводу-выводу. Кроме того, некоторые алгоритмы являются более естественными или эффективными и понятными (особенно при портировании их из императивных языков) при наличии присваивания и глобальных переменных.

Но, в общем, функциональное (и декларативное в целом) программирование радикально отличается от императивного программирования именно тем, что по сути функциональная программа представляет собой чистую математическую функцию, зависящую строго от своих аргументов, а выполнением программы становится вычисление для конкретных аргументов (или *аппликация*, т.е. применение, поэтому иногда говорят об *аппликативном программировании*).

Поэтому, обычно, в чисто функциональных языках программирования нет глобальных

переменных, нет оператора присваивания, или их действия носят строго ограниченный характер.

Давайте глянем на простой пример, который может показать разницу подходов при решении типовой задачи в императивном и функциональном программировании.

Например, мы хотим вычислить сумму кубов целых чисел от 1 до 100.

На JavaScript:

```
function f(x){
  return (Math.pow(x,3))
};

sum = 0;
for (i=1;i<=100;i++) {
  sum += f(i)
};
console.log("sum=" + sum);
```

На Haskell:

```
print $ sum $ map (^3) [1..100]
```

Разница в том, что для императивного языка мы явно задаем что и как делать, а для декларативного — просто пользуемся описательными методами.

К слову, на многих современных языках программирования, на том же JavaScript, есть возможность воспользоваться **map**:

```
function f(x){
  return (Math.pow(x,3))
};

var arr = [1,2,3,4,5,6,7,8,9,10];
console.log(arr.map(f));
```

если только получить список кубов, а вот полное решение с использованием ещё и метода reduce:

```
function f(x){
  return (Math.pow(x,3))
};

var arr = [1,2,3,4,5,6,7,8,9,10];
console.log( (arr.map(f)).reduce(function(sum,current){
  return sum + current}) );
```

Вот ещё вариант решения на Haskell:

```
print $ sum [x^3 | x <- [1..100]]
```

Правая часть записи абсолютно и неслучайно похожа на математическую нотацию:

$$\{x^3 \mid x \in \{1, \ldots, 100\}\}.$$