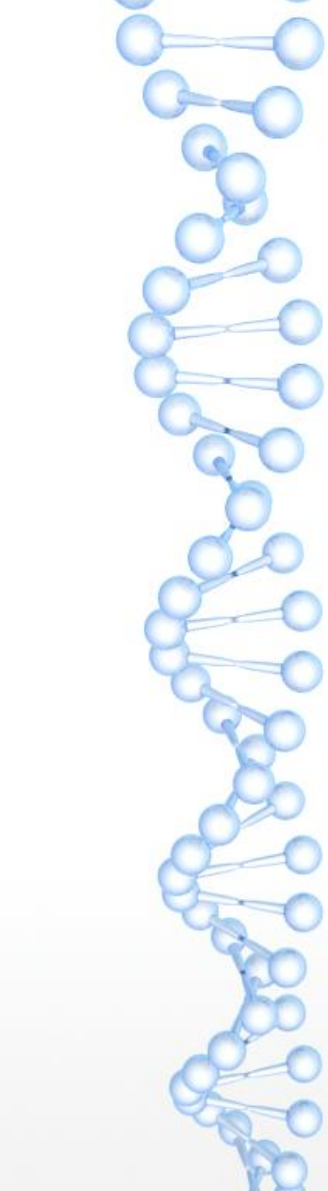
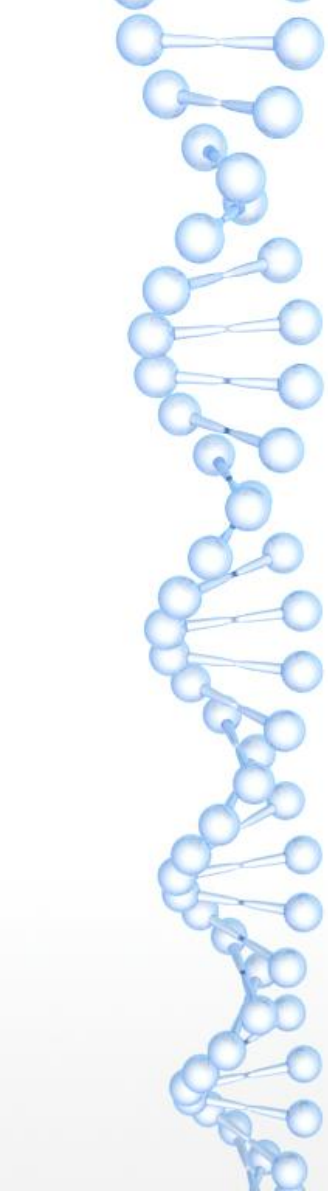


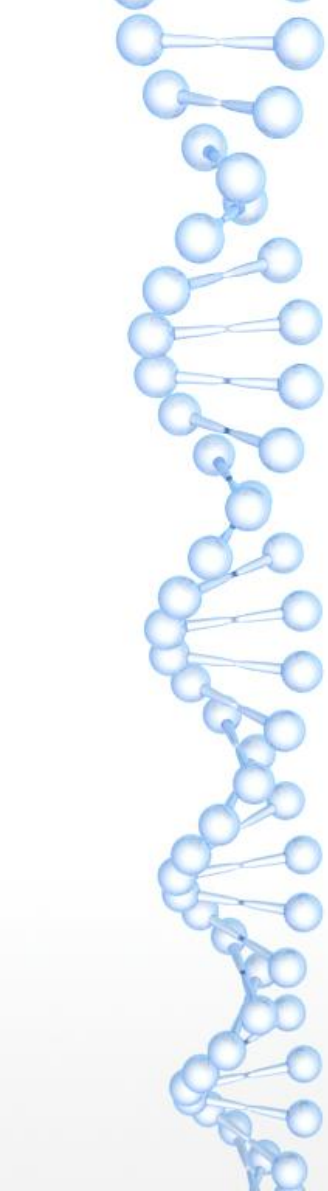
Функциональное и логическое программирование

Лекция 1. Введение в функциональное программирование



Учебные вопросы

- Структура дисциплины
- Принципы и достоинства функционального программирования
- Принципы лямбда нотации. Понятие лямбда терма
- Свободные и связанные переменные. Подстановка и преобразования



Принципы и достоинства функционального программирования

Определение парадигмы

Определение

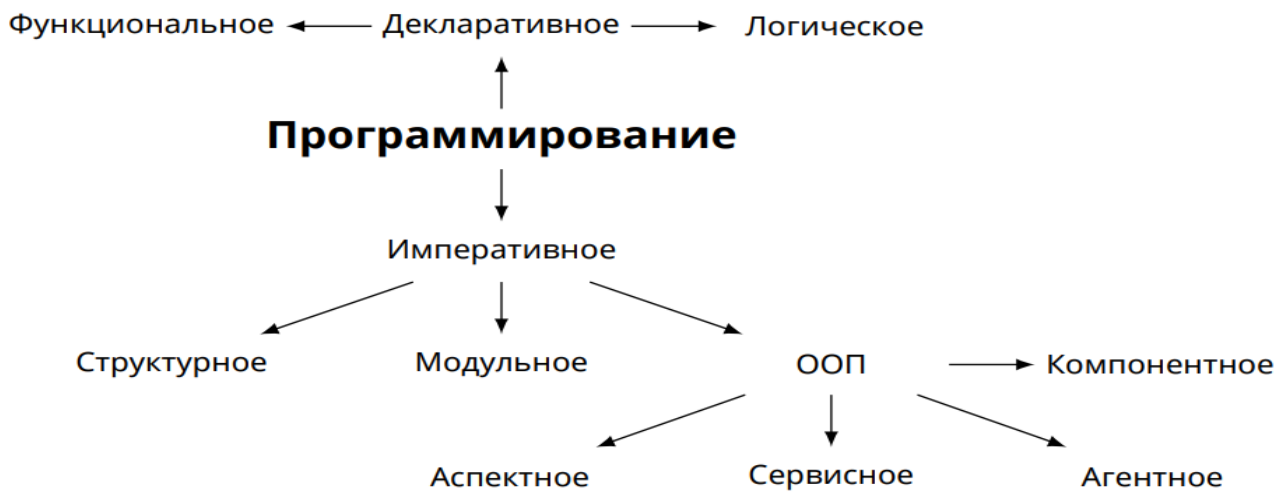
Парадигма программирования (англ. *programming paradigm*) — совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, построения их структуры и отдельных элементов программной системы.

Цель парадигмы программирования:

- разделение программы на базовые составные элементы (напр., функции или объекты);
- определение модели преобразования данных;
- внедрение ограничений на используемые конструкции.

Принципы и достоинства функционального программирования

Классификация парадигм программирования



Принципы и достоинства функционального программирования

Императивное программирование

Определение

Императивное программирование (англ. *imperative programming*) — парадигма, согласно которой программа представляет собой последовательность действий, изменяющих *состояние программы*.

Определение

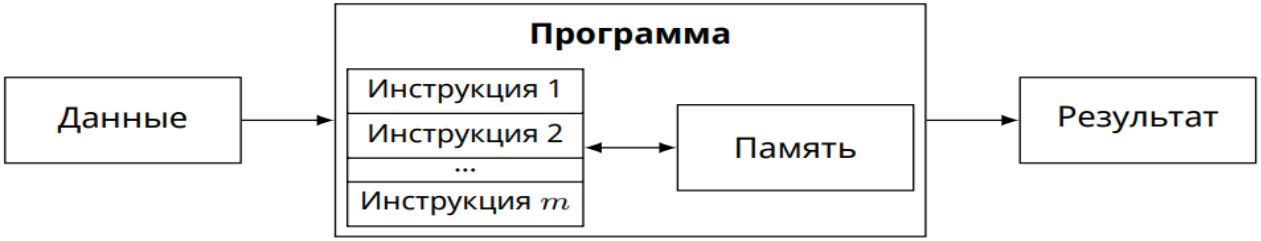
Состояние программы (англ. *program state*) — совокупность данных, связанных со всеми используемыми программой переменными в конкретный момент времени.

Область использования:

- системные программы;
- прикладные программы.

Принципы и достоинства функционального программирования

Выполнение императивной программы



Императивная программа использует именованные области памяти (переменные) для хранения состояния вычислений

Принципы и достоинства функционального программирования

Декларативное программирование

Определение

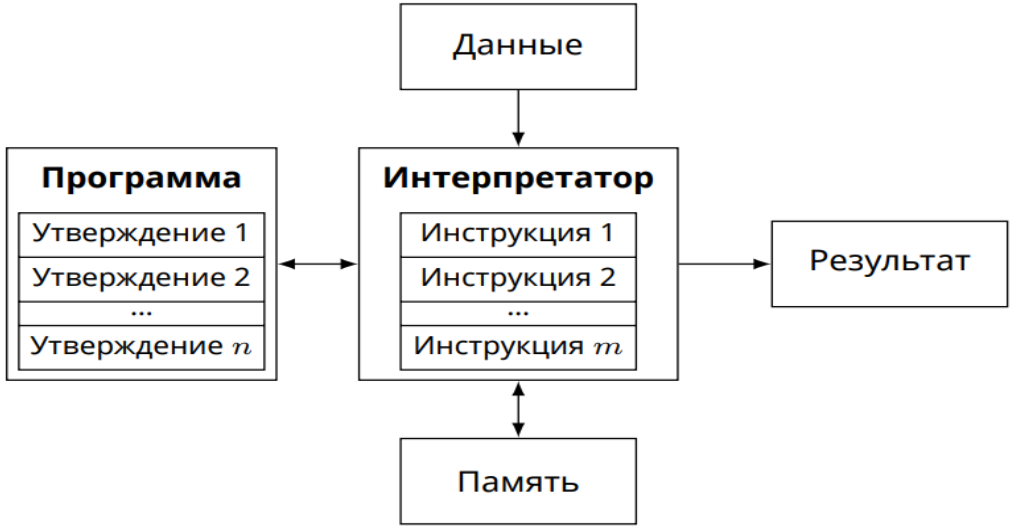
Декларативное программирование (англ. *declarative programming*) — парадигма, согласно которой программа представляет логику вычислений без описания прямой последовательности действий (действия определяются компилятором или интерпретатором).

Области использования:

- математическое моделирование;
- искусственный интеллект;
- анализ данных;
- наука.

Принципы и достоинства функционального программирования

Выполнение декларативной программы



Декларативная программа не взаимодействует напрямую с памятью, поручая эту работу интерпретатору

Принципы и достоинства функционального программирования

Функциональное программирование

Определение

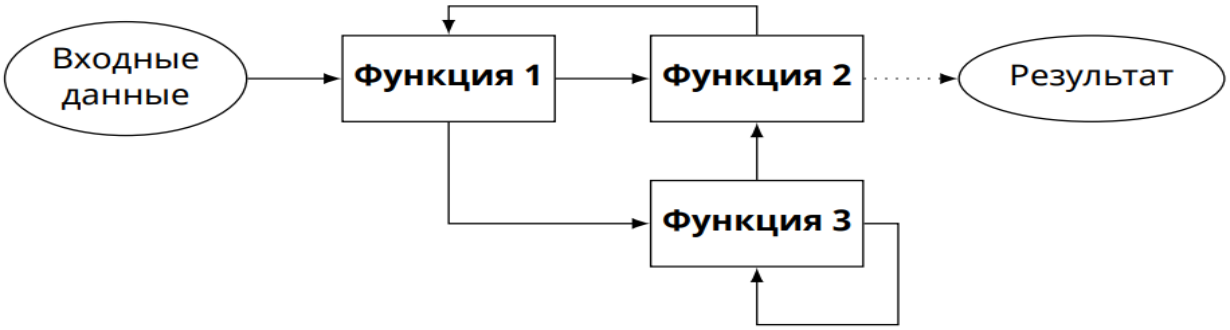
Функциональное программирование (англ. *functional programming*) — парадигма, согласно которой процесс исполнения программы представляется последовательностью вычислений значений для математических функций.

Особенности:

- отказ от явного хранения переменных (функции без побочных эффектов);
- $\Rightarrow$ встроенная поддержка параллелизации, оптимизации и кэширования без необходимости действий со стороны программиста.

Принципы и достоинства функционального программирования

Функциональное программирование (продолжение)



Функции возвращают результат, иначе не меняя состояние программы (напр., через переменные).

Ключевые слова: чистая функция, прозрачность ссылок.

Принципы и достоинства функционального программирования

Функциональное программирование (продолжение)

Концепции:

- функции высших порядков — функции, которые возвращают другие функции или принимают функции в качестве аргументов;
- замыкание (англ. *closure*) —сохранение контекста функции при ее создании;
- рекурсия для создания циклов;
- ленивые вычисления (англ. *lazy evaluation*) — вычисление аргументов функций по мере необходимости (не при задании).

Языки программирования: Lisp, Scheme, Clojure, Erlang, Haskell, F#.

Принципы и достоинства функционального программирования

```
let rec fibonacci n =
    match n with
    | 1 | 2 -> 1
    | _ -> fibonacci (n-1) + fibonacci (n-2)

let rec printFib n  =
    match n with
    | 1 -> printf "%d, " (fibonacci (n))
    | _ -> printFib (n-1)
           printf "%d, " (fibonacci (n))

printFib(16)
printfn "..."
```

Принципы лямбда нотации. Понятие лямбда терма

Основой λ-исчисления служит формальное понятие λ-термов, которые строятся из переменных и некоторого фиксированного множества констант при помощи операций применения (аппликации) функций и λ-абстракции. Это значит, что всевозможные λ-термы разбиваются на четыре класса:

1. **Переменные:** обозначаются произвольными алфавитно-цифровыми строками; как правило, мы будем использовать в качестве имён буквы, расположенные ближе к концу латинского алфавита, например, x, y и z.

2. **Константы:** количество констант определяется конкретной λ-нотацией, иногда их нет вовсе. Мы будем также обозначать их алфавитно-цифровыми строками, как и переменные, отличая друг от друга по контексту.

3. **Комбинации:** применение функции s к аргументу t, где s и t представляют собой произвольные термы. Будем обозначать комбинации как $s\ t$, а их составные части называть «ратор» и «ранд» соответственно.[2]

4. **Абстракция** произвольного λ-терма s по переменной x (которая может как входить свободно в s, так и нет) имеет вид $\lambda x.\ s$.

Свободные и связанные переменные. Подстановка и преобразования

В этом разделе мы формализуем интуитивное понятие свободных и связанных переменных, которое, между прочим, служит хорошим примером определения примитивно-рекурсивных функций. Интуитивно, вхождение переменной в заданный терм считается свободным, если оно не лежит в области действия соответствующей абстракции. Обозначим множество свободных переменных в терме s через $FV(s)$ и дадим его рекурсивное определение:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(c) &= \emptyset \\ FV(s\,t) &= FV(s) \cup FV(t) \\ FV(\lambda x.\, s) &= FV(s) - \{x\} \end{aligned}$$

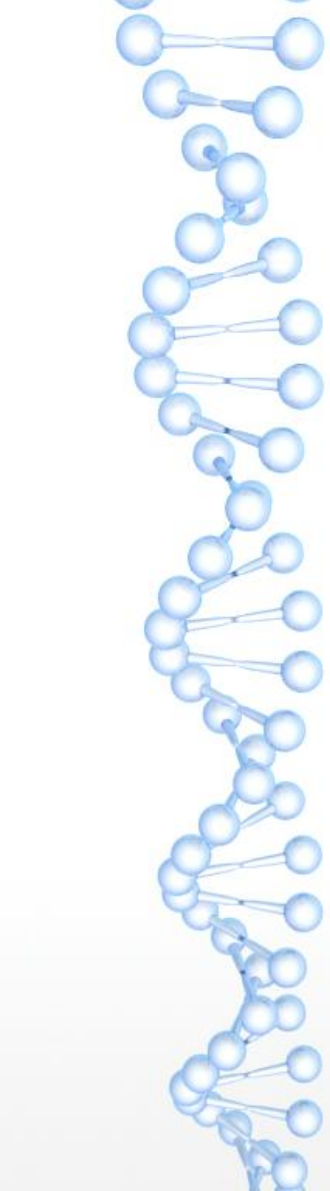
Аналогично вводится и понятие множества связанных переменных $BV(s)$:

$$\begin{aligned} BV(x) &= \emptyset \\ BV(c) &= \emptyset \\ BV(s\,t) &= BV(s) \cup BV(t) \\ BV(\lambda x.\, s) &= BV(s) \cup \{x\} \end{aligned}$$

Например, если $s = (\lambda x\ y.x)\ (\lambda x.z\ x)$, то $FV(s) = \{z\}$ и $BV(s) = \{x, y\}$. Отметим, что в общем случае переменная может быть одновременно и свободной, и связанной в одном и том же терме, как это было показано выше. Воспользуемся структурной индукцией, чтобы продемонстрировать доказательство утверждений о свойствах λ-термов на примере следующей теоремы (аналогичные рассуждения применимы и ко множеству BV).

Свободные и связанные переменные. Подстановка и преобразования

Теорема 2.1 *Для произвольного λ-терма s множество $FV(s)$ конечно.*
Доказательство: *Применим структурную индукцию. Очевидно, что для терма s, имеющего вид переменной либо константы, множество $FV(s)$ конечно по определению (содержит единственный элемент либо пусто соответственно). Если терм s представляет собой комбинацию t и, то согласно индуктивному предположению, как $FV(t)$, так и $FV(u)$ конечны, в силу чего $FV(s) = FV(t) \cup FV(u)$ также конечно, как объединение двух конечных множеств. Наконец, если s имеет форму $\lambda x.\, t$, то по определению $FV(s) = FV(t) - \{x\}$, а $FV(t)$ конечно по индуктивному предположению, откуда следует, что $FV(s)$ также конечно, поскольку его мощность не может превышать мощности $FV(t)$.*$\square$



Свободные и связанные переменные. Подстановка и преобразования

Одним из правил, которые мы хотим формализовать, является соглашение о том, что λ-абстракция и применение функции представляют собой взаимно обратные операции. То есть, если мы возьмём терм $\lambda x.\ s$ и применим его как функцию к терму-аргументу t, результатом будет терм s, в котором все свободные вхождения переменной x заменены термом t. Для большей наглядности это действие принято обозначать $\lambda x.\ s[x]$ и $s[t]$ соответственно.

Однако, простое на первый взгляд понятие подстановки одного терма вместо переменной в другой терм на самом деле оказалось весьма коварным, так что даже некоторые выдающиеся логики не избежали ложных утверждений относительно его свойств. Подобный грустный опыт разочаровывает довольно сильно, ведь как мы говорили ранее, одним из привлекательных свойств формальных правил служит возможность их чисто механического применения.

Обозначим операцию подстановки терма s вместо переменной x в другой терм t как $t[s/x]$. Иногда можно встретить другие обозначения, например, $t[x{:=}s]$, $[s/x]t$, или даже $t[x/s]$. Мы полагаем, что предложенный нами вариант легче всего запомнить по аналогии с умножением дробей: $x[t/x] = t$. На первый взгляд, рекуррентное определение понятия подстановки выглядит так:

Свободные и связанные переменные. Подстановка и преобразования

$$
\begin{aligned}
x[t/x] &= t \\
y[t/x] &= y, \text{ если } x \neq y \\
c[t/x] &= c \\
(s_1\ s_2)[t/x] &= s_1[t/x]\ s_2[t/x] \\
(\lambda x.\ s)[t/x] &= \lambda x.\ s \\
(\lambda y.\ s)[t/x] &= \lambda y.\ (s[t/x]), \text{ если } x \neq y
\end{aligned}
$$

Свободные и связанные переменные. Подстановка и преобразования

К сожалению, это определение не совсем верно. Например, подстановка $(\lambda y.\, x + y)[y/x] = \lambda y.y+y$ не соответствует интуитивным ожиданиям от её результата.[3] Исходный λ-терм интерпретировался как функция, прибавляющая x к своему аргументу, так что после подстановки мы ожидали получить функцию, которая прибавляет y, а на деле получили функцию, которая свой аргумент удваивает. Источником проблемы служит *захват* переменной y, которую мы подставляем, операцией $\lambda y.\ \ldots$, которая связывает одноимённую переменную. Чтобы этого не произошло, связанную переменную требуется предварительно переименовать:

$$(\lambda y.\, x + y) = (\lambda w.\, x + w),$$

а лишь затем производить подстановку:

$$(\lambda w.\, x + w)[y/x] = \lambda w.\, y + w$$

Свободные и связанные переменные. Подстановка и преобразования

$$
\begin{aligned}
x[t/x] &= t \\
y[t/x] &= y, \text{ если } x \neq y \\
c[t/x] &= c \\
(s_1\ s_2)[t/x] &= s_1[t/x]\ s_2[t/x] \\
(\lambda x.\ s)[t/x] &= \lambda x.\ s \\
(\lambda y.\ s)[t/x] &= \lambda y.\ (s[t/x]), \text{ если } x \neq y \text{ и либо } x \notin FV(s), \text{ либо } y \notin FV(t) \\
(\lambda y.\ s)[t/x] &= \lambda z.\ (s[z/y][t/x]) \text{ в противном случае, причём } z \notin FV(s) \cup FV(t)
\end{aligned}
$$

Свободные и связанные переменные. Подстановка и преобразования

- Альфа-преобразование: $\lambda x.\ s \underset{\alpha}{\longrightarrow} \lambda y.\ s[y/x]$, при условии, что $y \notin FV(s)$. Например, $\lambda u.\ u\ v \underset{\alpha}{\longrightarrow} \lambda w.\ w\ v$, но $\lambda u.\ u\ v \not\underset{\alpha}{\longrightarrow} \lambda v.\ v\ v$. Такое ограничение устраняет возможность ещё одного случая захвата переменной.
- Бета-преобразование: $(\lambda x.\ s)\ t \underset{\beta}{\longrightarrow} s[t/x]$.
- Эта-преобразование: $\lambda x.\ t\ x \underset{\eta}{\longrightarrow} t$, если $x \notin FV(t)$. Например, $\lambda u.\ v\ u \underset{\eta}{\longrightarrow} v$, но $\lambda u.\ u\ u \not\underset{\eta}{\longrightarrow} u$.