

Обзор парадигм программирования

Юрий Литвинов
yurii.litvinov@gmail.com

30.11.2018

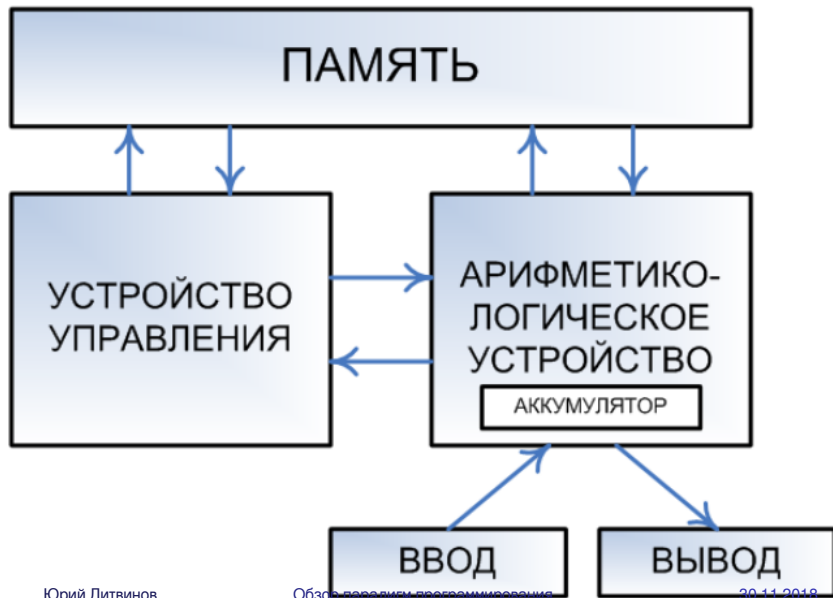
План

1. Структурное программирование
2. Объектно-ориентированное программирование
3. Функциональное программирование
4. Логическое программирование
5. Стековое программирование

Математические модели вычислений

- ▶ Что можно посчитать имея вычислительную машину неограниченной мощности?
- ▶ Формальные модели вычислений:
 - ▶ Машина Тьюринга
 - ▶ λ -исчисление Чёрча
- ▶ Тезис Чёрча: “Любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга.”

Архитектура фон Неймана



1. Структурное программирование

- ▶ Пришло на смену неструктурированному программированию в начале 70-х
- ▶ Любая программа может быть представлена как комбинация
 - ▶ последовательно исполняемых операторов
 - ▶ ветвлений
 - ▶ итераций
- ▶ Статья Дейкстры “Go To Statement Considered Harmful” (1968г)

Языки-представители

- ▶ Алгол
- ▶ Паскаль
- ▶ С
- ▶ Модула-2
- ▶ Ада

Подробнее: Ада

- ▶ Разработан в начале 80-х по заказу минобороны США
- ▶ Особенности:
 - ▶ Строгая типизация
 - ▶ Минимум автоматических преобразований типов
 - ▶ Встроенная поддержка параллелизма
- ▶ Реализация: GNAT <https://www.gnu.org/software/gnat/>

2. Объектно-ориентированное программирование

- ▶ Первый ОО-язык — Симула-67, были и более ранние разработки
- ▶ Популярной методология стала только в середине 90-х
- ▶ Развитие связано с широким распространением графических интерфейсов и компьютерных игр

Основные концепции

- ▶ Программа представляет собой набор объектов
- ▶ Объекты взаимодействуют путём отправки сообщений по строго определённым интерфейсам
- ▶ Объекты имеют своё состояние и поведение
- ▶ Каждый объект является экземпляром некоего класса

Основные концепции (инкапсуляция)

- ▶ Инкапсуляция — сокрытие реализации от пользователя. Пользователь может взаимодействовать с объектом только через интерфейс.
- ▶ Позволяет менять реализацию объекта, не модифицируя код, который этот объект использует

Основные концепции (наследование)

- ▶ 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

Основные концепции (полиморфизм)

- ▶ 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)

Пример кода

```
class Animal
{
    public:
        Animal(const string& _name) { name = _name; }
        virtual string talk() = 0;
        void rename(string newName);
    private:
        string name;
};
```

Пример кода (2)

```
class Cat : public Animal
{
    public:
        Cat(const string& name) : Animal(name) {}
        string talk() override { return "Meow!"; }
};

class Dog : public Animal
{
    public:
        Dog(const string& name) : Animal(name) {}
        string talk() { return "Arf! Arf!"; }
};
```

Языки-представители

- ▶ Java
- ▶ C++
- ▶ Object Pascal / Delphi language
- ▶ Smalltalk

3. Функциональное программирование

- ▶ Вычисления рассматриваются как вычисления значения функций в математическом понимании (без побочных эффектов)
- ▶ Основано на λ -исчислении

λ -исчисление

- ▶ λ -исчисление основано на функциях $\lambda x. 2 * x + 1$ — функция $x \rightarrow 2 * x + 1$
- ▶ 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)

Языки-представители

- ▶ Лисп (List Processing)
- ▶ ML (OCaml)
- ▶ Haskell
- ▶ Erlang

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

Что даёт ФП?

- ▶ Тестирование
- ▶ Отладка
- ▶ Параллелизм
- ▶ Горячая замена кода
- ▶ Машинные доказательства
- ▶ Оптимизация
- ▶ Ленивые вычисления

Примеры 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]
```

F#, мерджсорт

```
let rec merge l r =  
    match (l, r) with  
    | ([], r) -> r  
    | (l, []) -> l  
    | (x::xs, y::ys) -> if (x < y) then x::(merge xs r) else y::(merge l ys)
```

```
let rec mergesort l =  
    match l with  
    | [] -> []  
    | x::[] -> l  
    | _ ->  
        let (left, right) = List.splitAt (List.length l / 2) l  
        let ls = mergesort left  
        let rs = mergesort right  
        merge ls rs
```

F#, бесконечная последовательность простых чисел

```
let isPrime number =  
    seq {2 .. sqrt(double number)}  
    |> Seq.exists (fun x -> number % x = 0)  
    |> not
```

```
let primeNumbers =  
    Seq.initInfinite (fun i -> i + 2)  
    |> Seq.filter isPrime
```

Логическое программирование

- ▶ Программа представляет собой набор фактов и правил, система сама строит решение с использованием правил логики
 - ▶ Использует логику предикатов как математическую формализацию
- ▶ Создавалось в 60-х для решения задач искусственного интеллекта и экспертных систем
 - ▶ Автоматическое доказательство теорем
- ▶ Могут использоваться разные стратегии доказательства
 - ▶ В общем случае, программа — это набор фактов и правил + стратегия вывода, которая управляет тем, как новые факты получаются из существующих
 - ▶ В формальной логике стратегия вывода обычно не важна, для компьютеров это критично
- ▶ Дедуктивные базы данных — хранят факты и правила вывода

Пролог

- ▶ Появился в 1972 г. как научная разработка
- ▶ Реализации:
 - ▶ SWI-Prolog (<http://www.swi-prolog.org/>)
 - ▶ Amzi Prolog (<http://www.amzi.com/>)
 - ▶ Turbo Prolog
- ▶ Использует метод резолюций – последовательно перебирая правила и факты, пытается подобрать такой набор переменных, которые бы им удовлетворяли
 - ▶ Пример:
 - ▶ `cat(tom)`
 - ▶ `?- cat(tom).`
Yes
 - ▶ `?- cat(X).`
X = tom

Пример программы

```
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).
```

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

Yes

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

Императивное программирование

```
?- write('Hello world!'), nl.
```

```
Hello world!
```

```
true.
```

```
program_optimized(Prog0, Prog) :-  
    optimization_pass_1(Prog0, Prog1),  
    optimization_pass_2(Prog1, Prog2),  
    optimization_pass_3(Prog2, Prog).
```

QSort

```
quicksort(Xs, Ys) :- quicksort_1(Xs, Ys, []).
```

```
quicksort_1([], Ys, Ys).
```

```
quicksort_1([X|Xs], Ys, Zs) :-  
    partition(Xs, X, Ms, Ns),  
    quicksort_1(Ns, Ws, Zs),  
    quicksort_1(Ms, Ys, [X|Ws]).
```

```
partition([K|L], X, M, [K|N]) :-
```

```
    X < K, !,
```

```
    partition(L, X, M, N).
```

```
partition([K|L], X, [K|M], N) :-
```

```
    partition(L, X, M, N).
```

```
partition([], _, [], []).
```

Рекурсивное программирование, РЕФАЛ

- ▶ РЕкурсивных Функций АЛгоритмический
 - ▶ В. Турчин, 1966г.
- ▶ Ориентирован на символьные вычисления
 - ▶ ИИ, перевод, манипуляции с формальными системами (лямбда-исчисление, например)
- ▶ Использует нормальные алгорифмы Маркова в качестве математической формализации
- ▶ Программа записывается в виде набора функций
 - ▶ Функция — упорядоченный набор предложений
 - ▶ Предложение состоит из шаблона и того, на что надо заменить шаблон
 - ▶ Выражения в угловых скобках (активные выражения)
 - ▶ Переменные
- ▶ Вычисление продолжается, пока в «поле зрения» Рефал-машины не окажется выражение без угловых скобок

Рефал, пример

Hello, world:

```
$ENTRY Go { = <Hello>;}  
Hello {  
  = <Prout 'Hello world'>;  
}
```

Палиндром:

```
Palindrom {  
  s.1 e.2 s.1 = <Palindrom e.2> ;  
  s.1 = True ;  
  = True;  
  e.1 = False ;  
}
```

Стековое программирование

- ▶ Язык Форт (Forth)
 - ▶ Разработан в 60-х Чарльзом Муром «для себя»
 - ▶ Был широко распространён для программирования встроенных систем и задач, естественным образом выражающихся в терминах стеков
 - ▶ Синтаксический анализ
 - ▶ Анализ естественных языков

Форт, подробнее

- ▶ 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 ;

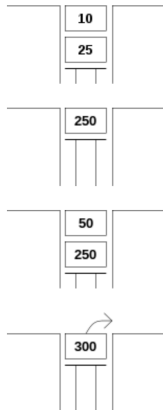
▶ то же самое на C:

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

▶ более красиво на Форте:

: FLOOR5 (n -- n') 1- 5 MAX ;

▶ : HELLO (--) CR ." Hello, world!" ;



Форт, пример

\ Напечатать знак числа

```
:.SIGN ( n -- )  
  ?DUP 0= IF  
    ." НОЛЬ"  
ELSE  
  0> IF  
    ." ПОЛОЖИТЕЛЬНОЕ ЧИСЛО" ELSE  
    ." ОТРИЦАТЕЛЬНОЕ ЧИСЛО" THEN  
THEN  
;
```

Реализации

- ▶ SwiftForth
 - ▶ <https://www.forth.com/swiffforth/>
- ▶ Gforth
 - ▶ <http://www.gnu.org/software/gforth/>
- ▶ Десятки других реализаций
 - ▶ <http://www.forth.org/commercial.html>
- ▶ Книжка
 - ▶ Броуди Л. «Начальный курс программирования на Форте»

