



Санкт-Петербургский государственный университет
Кафедра системного программирования

Лекция “Парадигмы программирования”

Юрий Литвинов
yurii.litvinov@gmail.com

Санкт-Петербург
2020

О чём пойдёт речь

- Опыт кафедры системного программирования СПбГУ в подготовке образовательного контента
- Пример (сокращённый) лекции для студентов 1-го курса бакалавриата “Технологии программирования”
- Обсуждение места лекции в курсе и программе в целом, опыт прочтения
- Инструменты для разработки материалов

Математические модели вычислений

- Что можно посчитать имея вычислительную машину неограниченной мощности?
- Формальные модели вычислений:
 - ▶ Машина Тьюринга
 - ▶ λ -исчисление Чёрча
 - ▶ Нормальные алгорифмы Маркова
- Тезис Чёрча: «Любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга»

Машина Тьюринга

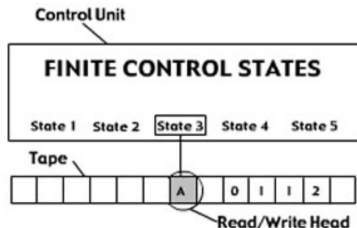
- Формально,

$$M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$$

$$\delta : (Q/F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

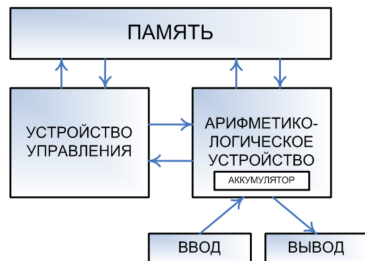
- Неформально:

- Бесконечная лента с символами из Σ и b
- Считывающая головка
- Внутренняя память Q
- Таблица переходов δ , которая по текущему состоянию из Q и текущему символу на ленте из Γ говорит машине, что делать:
 - ★ перейти в состояние
 - ★ записать символ на ленту
 - ★ сместиться влево/вправо



Архитектура фон Неймана

- Принцип последовательного программного управления
- Принцип однородности памяти
- Принцип адресуемости памяти
- Принцип двоичного кодирования
- Принцип жесткости архитектуры



Структурное программирование

- Пришло на смену неструктурированному программированию в начале 70-х
 - ▶ FORTRAN — 1957 год, язык высокого уровня, но не структурный
- 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.adacore.com/community>)

```
with Ada.Text_IO;
```

```
use Ada.Text_IO;
```

```
procedure Main is  
begin
```

```
  Put_Line ("Hello World");
```

```
end Main;
```


Ада, модульная система

```
package types is
  type Type_1 is private;
  type Type_2 is private;
  type Type_3 is private;
  procedure P(X: Type_1);
  ...
private
  procedure Q(Y: Type_1);
  type Type_1 is new Integer range 1 .. 1000;
  type Type_2 is array (Integer range 1 .. 1000) of Integer;
  type Type_3 is record
    A, B: Integer;
  end record;
end Types;
```

Ада, многопоточность и рандеву

```
with Ada.Text_IO; use Ada.Text_IO;
```

```
procedure Main is
```

```
  task After is
```

```
    entry Go(Text: String);
```

```
  end After;
```

```
  task body After is
```

```
  begin
```

```
    accept Go(Text: String) do
```

```
      Put_Line("After: " & Text);
```

```
    end Go;
```

```
  end After;
```

```
begin
```

```
  Put_Line("Before");
```

```
  After.Go("Main");
```

```
end;
```

Ада, ограничения и контракты

```
type Not_Null is new Integer
```

```
  with Dynamic_Predicate => Not_Null /= 0;
```

```
type Even is new Integer
```

```
  with Dynamic_Predicate => Even mod 2 = 0;
```

```
function Divide (Left, Right : Float) return Float
```

```
  with Pre => Right /= 0.0,
```

```
       Post => Divide'Result * Right < Left + 0.0001
```

```
       and then Divide'Result * Right > Left - 0.0001;
```

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
 - ▶ Принцип подстановки Барбары Лисков

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) {
```

```
            this.name = name;
```

```
        }
```

```
        void rename(const string &newName) {
```

```
            name = newName;
```

```
        }
```

```
        virtual string talk() = 0;
```

```
    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() override { return "Arf! Arf!"; }
};
```

Пример кода (3)

```
...  
Cat *cat1 = new Cat("Барсик");  
Animal *cat2 = new Cat("Шаверма");  
Dog *dog = new Dog("Бобик");  
  
std::vector<Animal *> animals{cat1, cat2, dog};  
  
for (Animal *animal : animals) {  
    std::cout << animal->talk();  
}  
...
```

- Java
- C#
- C++
- Object Pascal / Delphi Language
- Smalltalk

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

- λ-исчисление — формальный способ описать математические функции
 - ▶ $\lambda x. 2 * x + 1$ — 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)
- Формальная система, не требующая математических оснований
 - ▶ На самом деле, математика может быть построена на λ-исчислении

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

- Программы не имеют состояния и не имеют побочных эффектов
 - ▶ Нет переменных
 - ▶ Нет оператора присваивания
- Порядок вычислений не важен
- Циклы выражаются через рекурсию
- Ленивые вычисления
- Формальные преобразования программ по математическим законам

Пример 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#, 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
```

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 ;  
}
```

Место лекции в программе

- Лекция читается в середине первого семестра в курсе “Программирование”
- Сразу после контрольной работы
- Через неделю — вторая часть, с более экзотическими парадигмами (логическое, стековое, визуальное), часто Рефал отправляется во вторую часть
- Первый семестр — структурное программирование на Си, поэтому структурную парадигму можно подробно не рассматривать
- Второй семестр — объектно-ориентированное программирование на С#, поэтому ООП уделяется больше внимания
- Четвёртый семестр — функциональное программирование, к тому времени они всё забудут
 - ▶ Но, возможно, вспомнят, что это показалось им интересным

Место лекции в программе (2)

- Машины Тьюринга и формализация процесса вычислений — курс матлогики, 4-й семестр
 - ▶ Забудут детали, но вспомнят, что это зачем-то нужно и важно для программирования
- Углублённо функциональное программирование изучается на старших курсах, цель — заинтересовать
- Углублённо ООП с позиций архитектуры также на 4-м курсе, в “Проектирование ПО”
- Общая цель — расширение кругозора
 - ▶ Более конкретно, бороться с “эффектом утёнка” Си
 - ▶ Показать, что даже самих способов думать о программах много разных и что нельзя останавливаться на конкретном языке

Особенности чтения

- Неформальность изложения, цель — заинтересовать, а не научить
- Формальные определения даются, чтобы мотивировать изучать “математические” дисциплины
- Примеры по ООП в основном на C++, потому что он похож на Си
 - ▶ Уменьшается когнитивная нагрузка
 - ▶ Цель отчасти научить ООП, “first exposure”
- Примеры других парадигм на разных языках
 - ▶ Нет цели научить — цель заинтересовать, удивить, показать, что бывает

Опыт прочтения

- На лекции говорят, что всё понятно, в начале второго семестра ООП всё равно идёт тяжело
- Тем не менее, помогает при сдаче теории по информатике
- Студенты иногда считают, что лекция не очень вписывается в программу
- Без практики малополезно
 - ▶ В других образовательных программах есть целый курс “Парадигмы программирования”
 - ▶ Как мы предпочитаем преподавать парадигмы: ООП на программировании, по спецкурсу на каждую более экзотическую
- Иногда заражает сильных студентов энтузиазмом (особенно функциональное программирование или, далее, стековое)
- Более-менее обособленная лекция, можно рассказывать отдельно
 - ▶ Например, читалась как приглашённый доклад для школьников

Инструменты разработки лекции

- Презентация с примерами обязательна
- LaTeX, пакет Beamer
- Пакет minted для вёрстки кода
- Конспект желателен, но довольно дорог в разработке
 - ▶ Делается один раз, далее актуализируется каждое прочтение за гораздо меньшее время
- Learning Management System — Blackboard, не принципиально
- Версионирование контента — открытый репозиторий на GitHub
 - ▶ <https://github.com/yurii-litvinov/courses>
 - ▶ Не страшно, что виден студентам (даже хорошо, но они обычно не смотрят)
 - ▶ Потенциальные проблемы с лицензированием и авторским правом
 - ★ В примере выше лицензии нет, следовательно переиспользование невозможно