# Основные и базовые ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

ЯВУ – языки высокого уровня

# Роберт Флойд

- Автор основы теорий анализа и верификации программ, ряда весьма эффективных методов обработки данных.
- «Можно наладить ясное обучение ряду таких семантических методов для всех уровней программного проекта, чтобы у обученных студентов хватало головы на решение полного спектра проблем, начинающихся от упрощённой учебной постановки задачи до непрерывно расширяющегося класса практических задач »

# Тьюринговскую лекцию посвятил:

- проблеме обучения программистов,

- влиянию парадигм программирования на успех программистских проектов и

- как они должны быть поддержаны в языках программирования

# «Искусство программирования включает в себя расширение репертуара используемых парадигм»

Рекурсивные сопрограммы (Fortran)

Недетерминизм

Макро-техника

Lisp - программы как данные

Обучение семантическим методам:

от **упрощённой** учебной постановки задачи

до непрерывно расширяющегося класса **практических** задач

# Парадигма

Парадигма программирования - это исходная концептуальная схема постановки проблем и их решения.

Парадигма является инструментом грамматического описания фактов, событий, явлений и процессов.

Парадигмальные конструкции могут не существовать  одновременно, но интуитивно объединяться в общее понятие.

# Параметры парадигмальной классификации

**Уровень абстрагирования понятий**
(ЯНУ, ЯВУ, ЯСВУ)

**Степень изученности решаемой задачи**
(ЛП, ФП, ООП, ИП)

**Границы организованности алгоритма решения задачи** (Последовательные, параллельные модели)

**Удобство обучения программистов**
(ФП, параллелизм, ИП; ЛП, ООП)

Мера структурироособности программной системы

Показатель отлажванности представления программы

Область работоспенности программного инструмента

Ранг жизнеспособности программного продукта

# Проявление и поддержка

- Эксплуатационная прагматика решаемых задач
- Жизненный цикл задач и программ
- Абстрактный синтаксис ЯП
- Абстрактная машина ЯП
- Реализационная прагматика СП
- Концептуальные языки - монопарадимальные
- Мультипарадигмальные языки

# Основные парадигмы - *производственное значение*

Уровни (ЯНУ, ЯВУ, ЯСВУ)

- Императивно-процедурное
- Функциональное
- Логическое
- Объектно-ориентированное

Параллельное включается в каждую из основных

# Фундаментальные парадигмы

## *- образовательное значение*

*Базовые:*

- Функциональное
- Параллельное
- Императивно-процедурное

*Дополнительное расширение:*

- Логическое
- Объектно-ориентированное

# Факторы:

## *Степень изученности постановок задач*

- Макетный образец — концептуальный минимум.
- Экспериментальный полигон — потенциальный максимум.
- Практичная версия — производственный компромисс.
- Эффективная реализация — предельный баланс

# Факторы:
## *Уровни абстрагирования решений*

- **Низкоуровневое** программирование - возможность реализации эффективных решений ценой использования общего доступа к слабо защищенным структурам данных.

- **Языкам высокого уровня** - использование иерархии, независимые компоненты которой защищены от бесконтрольного взаимодействия.

- **Средства сверх высокого уровня** (языки спецификаций, языки параллельного программирования, системы представления знаний и т.д.) - полнота пространства решений по отдельным направлениям проблем, связанных с разработкой  и применением программ.

# Факторы:
## *Развитие технологий программирования*

- Создание языка Фортран сопровождалось появлением раздельной компиляции и доминированием парадигмы императивного программирования.
- Язык Лисп дал жизнь технологии символьных вычислений и парадигме функционального программирования
- Разработка языка Си как средства реализации операционных систем привела к технологии машинно-зависимого переноса программ и признанию ООП.

# Факторы:

## *Исследование и обучение программированию*

- Паскаль позволил сформулировать принципы структурного программирования как доминирующей парадигмы учебного программирования.
- Prolog показал возможности парадигмы логического программирования на базе частичного определения знаний.
- Идея организации процессов в языке Simula 67 и реализация работы с объектами в языке SmallTalk 80 дали импульс включения ООП в другие языки.

# СОВРЕМЕННАЯ СИСТЕМА ПРОГРАММИРОВАНИЯ

ЛАКОНИЗМ

КОНСТРУКТИВНОСТЬ

РАСШИРЯЕМОСТЬ

# Сравнение:
## *Операционная семантика СП*

- Лексически близкие ЯП могут привлекать удобочитаемостью на первых шагах отладки программ
- Синтаксически подобные ЯП могут быть удобны при подготовке и преобразовании программ при их переносе на другие системы
- ЯП с общим абстрактным синтаксисом семантически эквивалентны, они сравнимы по трудоёмкости отладки программ

# Сравнение:
## *Реализационная прагматика СП*

- ЯП с общей абстрактной машиной семантически равномощны, на их основе достижима сравнимая эффективность процессов вычислений.
- Диалекты ЯП часто бывают реализационно равнозначны, возможно семантически эквиваленты и равномощны, но могут быть лексически и синтаксически различимы

# Сравнение:
## *Варианты структур данных в памяти*

- Векторы с паспортом, хранящим при компиляции сведения о размерах и типе элементов.
- Запись или структура, обеспечивающая доступ к заданному перечню разносортных элементов по статически определённым ключам.
- Объединение заданного перечня разных ТД, размещаемых в разное время по определённому адресу.
- Множество небольшого числа перечислимых элементов, обработки которых не выводит за пределы машинных команд над битовыми кодами.

# Сравнение:
## *Методы доступа к структурам данных*

- New/delete — динамические запросы к системе памяти типа «куча»;
- Компактизация — уплотнение пространства с целью размещения крупных целостных объектов;
- «Близнецы» — метод укрупнения памяти и профилактики её чрезмерной фрагментации при распределении на разно объёмные блоки;
- Стек — совмещает две дисциплины доступа (FILO и вектор).
- SetI — более 17-ти разных СД, динамически выбираемых СП для представления множеств в зависимости от характера их обработки и наличия свободной памяти.

# Достоинства структурного программирования

- методологически доминирующая парадигма надёжного программирования,
- нацеленность на нисходящее проектирование,
- пошаговое улучшение программы решения,
- сведение задачи к более простым подзадачам,
- переход от конкретных объектов и функций машинного уровня к более абстрактным объектам и функциям, позволяющим продумывать модули, выделяемые при нисходящем проектировании,
- чёткая иерархия определений и управления с однозначно определённым типом результатов в точках схождения.

# ЯВУ
# Поляризация характеристик программы

Программируемые решения представляются в **императивно-процедурной** или в *декларативной форме.*

Программа может быть **защищена от изменений в процессе её выполнения** или *допускать программируемые модификации по ходу получения результатов вычисления.*

Программа может **быть целостной** или *собираться из типовых компонент и шаблонов.*

*Домены обрабатываемых значений могут быть* ***независимыми*** *или допускать пересечения.*

# ЯВУ
# Поляризация характеристик данных

Обрабатываемые элементы данных **позиционируются как адресуемые блоки памяти** или *независимо размещаемые значения.*

Распределение и повторное использование памяти может **быть действием в программе** или *выполняться автоматически системой программирования.*

Результат выполнения программы может быть **рассредоточен по ряду переменных** или *сконцентрирован в специальном регистре.*

# ЯВУ
# Поляризация характеристик памяти

Области видимости имён могут **быть глобальными** или локализованными по иерархии конструкций с возможностью восстановления контекста.

Инициирование памяти первоначально размещаемыми значениями **может требовать программируемых действий** или *выполняться в по умолчанию.*.

Контроль правильности может выполняться **статически – при анализе текста программы** или *динамически – при выполнении кода программы.*

# ЯВУ: Поляризация характеристик *управления вычислениями*

Представленные в программе функции могут **быть типизированными** или *универсальными, дающими разумную реакцию на любой элемент данных.*

*Управление вычислениями выполняется* ***последовательно*** *или параллельно,* порядок действий может быть **определённым** или *недетерминированным.*

Вычисления могут быть **энергичными** или *ленивыми.*

# ИП

## *Типичные схемы постановок задач*

**Определён алгоритм** решения
актуальной задачи.

Необходимо получить
**программу реализации алгоритма**
с практичными пространственно-временными характеристиками
на доступном оборудовании.

# ФП

## *Типичные схемы постановок задач*

Известна **предметная область**.

Следует выбрать **символьное**
представление данных для этой области
и отладить систему универсальных **функций**,
пригодных для создания **прототипов**
решения актуальных задач из этой области

# ЛП

## *Типичные схемы постановок задач*

Дана **коллекция фактов** и отношений,

характеризующая актуальную задачу.

Надо привести эту коллекцию к форме,
демонстрирующей вычислимость ответов

**на практичные запросы**

относительно данной задачи.

# ООП

## *Типичные схемы постановок задач*

Доступна иерархия классов объектов с **работоспособными методами решения** ряда задач некоторой сферы приложения ИТ.

Требуется **без лишних трудозатрат**

**уточнить** эту иерархию, чтобы приспособить её

**к решению новых востребованных задач**

этой области, её расширения или ей подобной.

# Первые!

А.А. Ляпунов - операторный метод для описания программ, постановка задачи автоматизации программирования

Л.В. Канторович - технология крупноблочного программирования

Гр. Хоппер (Grace Hopper) разработала первый компилятор

Дж. Бэкуса (John Bakus) - первый алгоритмический язык FORTRAN, БНФ, FP

К.Ю.Айверсон ( Kenneth Eugene Iverson)– первый язык параллельного программирования APL

# ЯЗЫКИ С БОЛЬШИМ ВЛИЯНИЕМ

Fortran – 54-57
Lisp – 58-60
APL – 57-62
Cobol – 59
PL/1 – 64
Snobol – 62-67
C – 72

Prolog - 74
C++ - 85
Haskell - 90
CLOS - 94
Java - 95
IL - 2003

# Идеи функционального программирования

Механизм *функций*

Джон Мак-Карти  - 1958

Реализация языка *LISP (LISt Processing)*

Символьная обработка данных

Алгебраические аппликативные системы

Лямбда-исчисление Чёрча - 1928

Искусственный интеллект

# Понятие *"функция"*

Идеальные чистые функции (pure)

Оттеснение присваиваний и передач управления на периферию

Независимость от контекста описаний

Выделение **чистого результата**

Без синтаксического разнообразия

Рекурсивные функции, отображения и категории

Специальные функции (макро и мульти)

Псевдо-функции (монады)

# Интерпретация:
## *неимперативное управление процессами*

Ленивые вычисления / отложенные

Энергичные вычисления / опережающие

Мемо-функции / таблицы

Недетерминизм

Параллельные процессы

Отладка / компиляция отлаженных функций

Спецификации и макеты=прототипы

# Реализация систем функционального программирования

Списки свойств атомов и структура списков

Представление структуры списка

Список свободной памяти и «сборщик мусора»

Сборка системы и ее рабочий цикл

Динамическая  память и структуры данных

Реальный состав системы и внешний мир

Деструктивные (разрушающие) преобразования

# Хаскелл Карри – http://haskell.org

**ЯП должен быть:**

- пригоден для обучения, исследований и приложений, включая построение больших систем,
- полностью описываться с помощью формальных синтаксиса и семантики,
- находиться в свободном доступе, следует разрешить свободную реализацию и распространение языка,
- базироваться на идеях, которые получат широкое одобрение,
- сократить излишнее многообразие языков функционального программирования.

# Повлияли:

Lisp ( и Common Lisp, и Scheme),
ISWIM (Landin),
APL,
FP (Backus),
ML и Standard ML,
Hope и Hope+,
Clean,
Id,
Gofer,
Sisal
Miranda (Turner)

# Быстрая сортировка (Sisal)

```
Function QS (Data)
  If  ( size (Data) < 2 then Data )
                              else
      Let Pivot :=  Data [ liml (Data)]
            Low, Mid, High := for E in Data do
                                    returns array of E when E < Pivot,
                                            array of E when E = Pivot,
                                            array of E when E > Pivot
                                  end for
      in QS (Low) || Mid || QS (High)
  end if
end function
```

# «Новые решения»
## или забытые старые

Haskell - предпочтение «ленивой» схемы вычислений и концепция «монад»

Sisal - участки с однократными присваиваниями и циклы с выделением позиций для пространства параллельных итераций, сборки параллельно полученных результатов и обработки потоков данных

Python - средство разработки распределенных систем и сетевого программирования

C# - возможность встраивать функциональные построения в контекст императивных программ

F# - возможность оперировать деревом разбора программы и ее исполнимым кодом

Спасибо за внимание!