1. История развития ЯП

20 гг 20 века – появление ВМ, работающей на перфокартах.

Ада Левлейс – разработка конструкций ЯП

Моучли – кодирование команд при помощи мат. Формул

Хоппер – понятие отладка, подпрограмма, компилятор

1954 – фортран – решение научных расчетов

Конец 50х – алгол. Алгоритмический язык, позволяющий писать алгоритм при помощи функций

Середина 60х – flow matic. В отличие от фортрана, предназначался для решения бизнес задач (обработка данных и тд). Развитие этого подхода вылилось в создание ЯП cobol

Конец 60х – появление языка Basic, содержащий простые английские слова в качестве синтаксиса

Конец 60х – попытка сделать универсальный язык PL\I. Далее эту идею поддержал алгол-68. Была идея, что такие языки будет развиваться и заменять друг друга. Не получилось

70е – Паскаль, Никлаус Вирт. Изначально учебный язык

1982 – появление С. Очень нравился системным программистам. Первое ядро ОС Юникс было написано на нем

Далее С ушел в разработку ANSI (урезанный формат). Полученная модель вышла в свет в 90е. На основе него построены современные Java и C++

90е – появление скриптовых ЯП: питон, раби, перл

2. Специализированные ЯП

Специализированный ЯП – ЯП, предназначенные для решения одной, максимум несколько, видов задач (работа с бд и тд)

Пример:

PL-SQL (Oracle)

Visual Basic for Application (Access)

3. Парадигмы программирования. +/-

Парадигма – набор идей и понятий, определяющий стиль написания проги.

*Структурное:*

- структура проги – иерархия

- вся прога разбивается на блоки: последовательность, цикл, ветвление

Плюсы:

- независимость от аппаратных средств: for он и на пеньке 2, и на райзоне 5 будет for

- если не нужно думать об архитектуре – увеличение скорости разработки программы

Минусы:

- необходимость выполнения приведения к архитектуре – снижение скорости выполнения программы

*Функциональное*

- программа – набор функций

- функция может использовать результаты других функций и являться результатом для другой функции

- запуск функции возможен только тогда, когда все значения в функции есть

Плюсы:

- динамическое распределение памяти

- возможность сосредоточиться на предметной области а не на рутинных задачах

Минусы:

- структура нелинейная – сложность в понимании написанной программы

*Логическое:*

- автоматическое доказательство теорем на базе заданных фактов и правил вывода

Плюсы:

- возможность откатов, возвращение к предыдущей подцели в случае отрицательного результата одного из вариантов в процессе поиска решения.

- нет необходимости в решении задачи методом полного перебора

Минусы:

- узкий круг решаемых задач

*Автоматное программирование*

- любой фрагмент программы – автомат

- в простом случае – ДКА, в непростом – НКА

*ООП*

- все представлено в виде объектов

- взаимодействие между объектами осуществляется по средствам методов

- методы описывают поведение

- обмен сообщений

Плюсы:

- наследование позволяет строить производные понятия на основе базовых

- наследование от базовых позволяет увеличить скорость разработки

- полиморфизм дает гибкость разработке

- удобно разрабатывать группой лиц

Минусы:

- сложность в формализации реального мира дает сложность в тестировании ПО

4. Императивное, декларативное. +\-

Императивное – говорим, что хотим сделать. То есть указываем последовательность действий.

Императивное программирование подразумевает:

- последовательность действий

- операторы присвваивания

- чтение операндов из памяти

- запись полученных результатов в память

Плюсы:

- просто написать

- быстро написать

Минусы:

- медленно

- ошибки, связанные с присваиванием

ЯП Джава

1. Общая характеристика