Языки программирования

Лектор Головин Игорь Геннадьевич

Доцент каф. алгоритмических языков

Определение ЯП

1). Экстенсиональное (через объем понятия)

С, Паскаль, Java, C#, Фортран, C++.....

Определение ЯП

1). Экстенсиональное (через объем понятия)

C, Паскаль, Java, C#, Фортран, C++.....

Сколько их? Тысячи...

Джин Саммет (1969 - 120), DoD (1975 -> 350, 1983->450

Википедия - 308 (ru), 686 (en)

Динозавры....

Но проблема - а что же не ЯП?

Определение ЯП

1). Экстенсиональное (через объем понятия)

C, Паскаль, Java, C#, Фортран, C++, JavaScript, Lua, Python.....

Сколько их? Тысячи (Дж.Саммет - 450 в 1967, >3000 в 1975)

Википедия - 308 (ru), 686 (en)

Но проблема - а что же не ЯП?

English, русский, эсперанто - нет!

SQL - да, а HTML? CSS? XSLT?XML? Язык утилиты Make?

Алгоритмическая полнота? - XSLT - да, SQL-92 - нет

Charity - ЯП для ФП - нет (зато заканчивается любая программа, написанная на нем)

Определение ЯП

2). Интенсиональное (через свойства)

В.Ш.Кауфман ("Языки программирования: концепции и примеры"):

язык программирования - это инструмент для планирования поведения исполнителя

Определение ЯП

2). Интенсиональное (через свойства)

В.Ш.Кауфман ("Языки программирования: концепции и примеры"):

язык программирования - это инструмент для планирования поведения исполнителя

Исполнитель -> управляемое ***автоматическое*** поведение

Программа -> Поведение

Форд Т - не наш исполнитель

Яндекс-мобиль - вполне.

Ткацкие станки Жаккарда (начало 19 века) - программа определяла текстуру ткани(Ч.Бэббидж использовал устройство ввода для своего устройства)

Определение ЯП

2). Интенсиональное (через свойства)

Исполнитель - браузер => связка HTML/CSS (+XSLT + XML +....) - ЯП, но ОЧЕНЬ специализированный.

Наш исполнитель - вычислительная система (компьютер + ОС)

Управление -> компьютерная программа

ЯП - формальная нотация для записи компьютерных программ

Нас будут интересовать не любые ЯП для написания программ, а те, которые обладают двумя свойствами:

-универсальность

-индустриальность

Виды программирования

- игровое (не путать с программированием игр)
- научное
- индустриальное

Главные критерии различия - круг пользователей программы (для кого пишем) и степень отчуждаемости

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

10>

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

10> LET H$ = 'Hello'

20>

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

```
10> LET H$ = 'Hello'
20> LET W$ = 'world!'
30> PRINT H$ & ',' & W$
```

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

```
10> LET H$ = 'Hello'
20> LET W$ = 'world!'
30> PRINT H$ & ',' & W$
Hello, world!
```

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

```
10> DIM A(10)
20> FOR I=1, 10
30> A(I) = I
40> NEXT I
```

MS DOS => BASICA.COM (или BIOS IBM PC)

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

Паскаль (1970)

Виды программирования

- игровое (не путать с программированием игр)

Для себя. Хобби, развлечение, учеба

Языки:

BASIC (1964) -**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

Паскаль (1970)

Малоприменимы для написания чего-то отличающегося от хобби/учебы

Повторное использование программ - практически нет

Б.Керниган: "16 причин, по которым Паскаль не является моим любимым языком программирования"

Виды программирования

- игровое (не путать с программированием игр)
- научное

Программа нужна для получения какого-то результата (как правило, вычислительного), который является значимым для определенного круга профессионалов/ученых

Повторное использование - относительно редко и, как правило, требует консультаций с автором (или же только автор)

Иногда подобные программы перерастают в промышленные, но требуют практически полного переписывания ("с нуля")

Виды программирования

- научное

Языки:

ФОРТРАН - 1954-1957 - Фортран I, далее Фортран II, Фортран IV, Фортран 66-стандарт, Фортран 77, Фортран 90, Фортран 95, Фортран 2003, Фортран 2008, Фортран 2018

FORmulae **TRAN**slator - руководитель проекта - Джон Бэкус (IBM)

Самый первый ЯП и в определенном смысле - самый удачный

Почему?

Виды программирования

- научное

Языки:

ФОРТРАН - **FOR**mulae **TRAN**slator

Самый первый ЯП и в определенном смысле - самый удачный

Почему?

1. Четко поставленные и достигнутые главные цели - заменить посредника между математиком и компьютером, обеспечить приемлемую эффективность вычислений (по сравнению с программой в машинных кодах или ЯА)
2. Дополнительные эффекты (неожиданные) - прежде всего "мобильность" знаний, которая была дополнена мобильностью программ, что привело к накоплению ОГРОМНОГО количества библиотек для научных вычислений
3. До сих пор самый эффективный язык для "number crunching" (конечно, компиляторы для специализированных архитектур)

Виды программирования

- научное

Языки:

ФОРТРАН - **FOR**mulae **TRAN**slator

В остальном - самый "плохой" и ругаемый (в старых книгах по ЯП) язык программирования

За что? (учтите, что критика относится к версиям до 77 года)

- только статические данные (все данные в статических сегментах - см. учебник по ЯА), следовательно отсутствие рекурсии и динамических струтур данных

- очень слабые и ненадежные межмодульные связи
- отсутствие структурных управляющих конструкций (4 вида оператора перехода , один оператор цикла и условный оператор, который по сути есть оператор перехода)
- единственная структура данных - массив
- единственный вид модуля - подпрограмма (функции и процедуры)
- архаичный способ записи (одно предложение языка - одна перфокарта (строка), форматированная запись по колонкам)
- общая ненадежность языка

Ну и еще много чего "хорошего" можно сказать...

Язык стал "контрпримером" с точки зрения дизайна языковых конструкций

Виды программирования

- научное

Языки:

ФОРТРАН - **FOR**mulae **TRAN**slator

Самый плохой и самый удачный ЯП одновременно!!!

Почему оба утверждения верны и не противоречат друг другу?

Виды программирования

- научное

Языки:

ФОРТРАН - **FOR**mulae **TRAN**slator

Самый плохой и самый удачный ЯП одновременно!!!

Почему оба утверждения верны и не противоречат друг другу?

Тут работает "принцип экологической ниши ЯП"

Экологическая ниша ЯП

Конечно же - аналогия с биологическим понятием

Экологическая ниша ЯП - это проблемная область (или набор проблемных областей), в которых применяется ЯП

Примеры проблемных областей:

- научные высокопроизводительные вычисления (мат.физика и т.п.)
- научные экспериментальные вычисления (то есть классическое научное программирование, но не требуется суперскорость)
- системное программирование (ОС, драйверы, низкоресурсные вычислительные устройства)
- бухгалтерия, финансы (на самом деле - множество ПО)
- веб-разработка на стороне клиента

Каждая из этих областей содержит множество более мелких

Экологическая ниша ЯП

Принцип экологической ниши ЯП: доминирующим языком в нише является тот, который занял эту нишу первым.

"Занятие ниши" означает то, что язык подходит для этой ниши, то есть пользователи языка (то есть программисты) активно используют ЯП для этой ниши.

Экологическая ниша ЯП

Принцип экологической ниши ЯП: доминирующим языком в нише является тот, который занял эту нишу первым.

"Занятие ниши" означает то, что язык подходит для этой ниши, то есть пользователи языка (то есть программисты) активно используют ЯП для этой ниши.

Заметим, что вопрос о том, какой язык "лучший", то есть лучше подходит для ниши, вообще не упоминается!

Вытеснение языка из ниши происходит при изменений условий в нише (появлении дополнительных подобластей, новых возможностей и т.д.), а вовсе не из-за того, что появился новый "ну очень суперский" язык программирования, на котором так здорово писать программы для этой проблемной области.

Именно поэтому Фортран до сих пор чувствует себя прекрасно в свой нише высокопроизводительных вычислений, в то время как выделившуюся нишу чисто экспериментальных вычислений он потихоньку уступает конкурентам (Python, R, Julia, Matlab)

Экологическая ниша ЯП

Проблемы Фортрана начались тогда, когда его стали использовать для совсем не подходящих для него ниш(системное программирование, управление устройствами и т.п.), то есть ниш, которые он не смог занять по причине своей "ненадежности".

Надежность языка характеризуется тем, насколько язык "провоцирует" (уменьшает надежность) или наоборот уменьшает

Пример из Фортрана (1964 год) - программа, управлявшая запуском ракеты к Венере

Фрагмент программы с оператором цикла, который прибавляет фиксированное значение к каждому элементу массива из 3 элементов (пример переделан, но смысл остался)

```
      DO 5 I = 1,3
      A(I) = A(I) + C
5     CONTINUE
```

Экологическая ниша ЯП

Это был фрагмент, который ДОЛЖЕН был быть.

А вот реальный фрагмент с ошибкой:

```
        DO 5 I = 1.3
        A(I) = A(I) + C
5       CONTINUE
```

Синтаксически получившийся фрагмент выглядит так:

do5i = 1.3 //это оператор присваивания

A(I) = A(I) + C // тоже оператор присваивания (I скорее всего уже

//использовалось перед этим фрагментом и имеет какое-то значение)

пустой оператор

Проблемы дизайна - неявное объявление переменных, лексика, синтаксис, отсутствие контроля диапазона индексов

Но с точки зрения главной ниши языка - мелочи...

А с точки зрения ниш, где надежность критична, - абсолютно неподходящий язык

Экологическая ниша ЯП

Другой пример прочно занятой ниши - системное программирование. Тут все ясно - это язык С. До него эту нишу занимал ЯА, но он оказался абсолютно неадекватным, что привело к появлению в 60-е годы ряда языков-претендентов (BLISS, АСТРА, АЛМО, BCPL и много других), но С (линия BCPL->B->C) сразу после своего появления оккупировал эту нишу. Новые языки-претенденты (Модула, Модула-2, Ада, Оберон) не имели шансов в этой проблемной области.

COBOL

Виды программирования

Вернемся с рассмотрению видов программирования и ЯП, которые соответствуют этим видам.

- научное

Языки:

ФОРТРАН - **FOR**mulae **TRAN**slator (1954-...)

АЛГОЛ (**ALGO**rithmic **L**anguage) - первая версия - 1958 год, окончательный вариант - 1960 - АЛГОЛ-60

Вообще не для программирования, а для обмена алгоритмами

С 1960 в журнале Communications of the ACM (CACM) алгоритмы публиковались на Алголе-60

Алгоритм 64 на языке Алгол 60 (неполный)

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  quicksort (A,M,N);  value M,N;
             array A;  integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin      integer I,J;
           if M < N then begin partition (A,M,N,I,J);
                               quicksort (A,M,J);
                               quicksort (A, I, N)
                         end
end        quicksort
```

Алгоритм 64 - быстрая сортировка (изображение взято с сайта https://www.cs.princeton.edu/courses/archive/spring20/cos226/lectures/23Quicksort.pdf)

Виды программирования

- научное

Языки:

ФОРТРАН - **FOR**mulae **TRAN**slator (1954-...)

АЛГОЛ (**ALGO**rithmic **L**anguage) (1958-60-68)

Сейчас ученые используют для экспериментов чаще всего следующие языки (и системы программирования):

Python, Julia, R, Matlab, Mathematica, MathCad и много других

Виды программирования

И наконец последняя разновидность (last but not least)

- игровое (не путать с программированием игр)
- научное
- **индустриальное**

Главные критерии различия - круг пользователей программы (для кого пишем) и степень отчуждаемости

Индустриальное программирование - создание программных продуктов, то есть программ и систем, которые обязательно отчуждаются, то есть используются независимо от автора

ЯП для индустриального программирования

Где используются индустриальные программы? ВЕЗДЕ

Огромное количество проблемных областей

Каждая проблемная область предъявляет свои требования к инструментам, главным из которых является система программирования, а в СП - ядро - это ЯП.

Такие требования будем называть ***технологическими потребностями***

ТП для разных ПО - могут противоречить друг другу

Примеры:

1) ПО - встроенное программное обеспечение, работающее в РВ для низкоресурсных устройств

2) Веб-программирование (например, сайт стартапа или интернет-магазина)

Разные требования, разные ТП, разные инструменты (ЯП)

1) C/C++, Ада
2) клиент(front-end) - JavaScript, сервер(back-end) - КУЧА фреймворков PHP/WordPress, Python/Django, Ruby On Rails, C#(VB)/Entity Framework, JavaScript/NodeJS/Express, ... далее десятки (если не сотни....) но не C/C++

Но! На чем написаны сами сервера (равно как и ОС, на которых эти сервера работают)?

Общие ТП - модульность, раздельная трансляция и др.

ЯП для индустриального программирования

Нас будут интересовать ТОЛЬКО ЯП для индустриального программирования

Что будем изучать?

1) Как ЯП отвечают наиболее общим ТП
2) Общие свойства ЯП, используемых в современной индустрии разработки ПО

ЯП и парадигмы программирования

Парадигма программирования - совокупность идей, методов, понятий, определяющих стиль программирования и способ мышления при создании программ.

Много мнений, определений и т.п.

Будем понимать максимально инклюзивно.

Только 3 "базисные" парадигмы:

- императивная (процедурная)
- функциональная
- логическая

Есть "подпарадигмы" - вкладываются или основываются на базисных

ЯП и парадигмы программирования

Подпарадигмы:

Пример 1:

объектно-ориентированная - основана на императивной =>

объектно-императивная парадигма

Объект <=> состояние + поведение

Объекты посылают друг другу сообщения. При обработке сообщений объект может менять свое состояние и посылать сообщения, меняя тем самым состояние других объектов.

Понятие состояния теснейшим образом связано с императивной парадигмой

ЯП и парадигмы программирования

Подпарадигмы:

Пример 2:

обобщенное программирование (generic programming)

Опять же много определений

Будем понимать "узко".

Статическая параметризация типов (реализация - родовые сегменты в Аде, шаблоны в C++, обобщения в C# и Java)

Отличие от примера 1 - может "обогащать" любую парадигму

Haskell - "чистый" функциональный ЯП со статической параметризацией типов

ЯП и парадигмы программирования

Зачем нам ПП?

Фундаментальное понятие => важно понимать концепции парадигм и то, как они отражаются в конкретных ЯП

Современные индустриальные ЯП - либо чисто императивные (C), либо содержат элементы других парадигм, но с доминированием одной.

В настоящее время - объектно-императивная парадигма

Включения из ФП, возможно использование статической параметризации

Сейчас - введение в ПП на примере простейшей задачи, решенных в разных стилях и разных языках.

Задача: реверс входного потока (символов)

А роза упала на лапу Азора =>  => арозА упал ан алапу азор А

Императивная парадигма

Основана на т.н. фон-неймановской модели

ОЗУ("память") <=> ЦПУ <=> ВЗУ ("ввод-вывод")

ОЗУ <=> программа + данные

ЦПУ <=> УУ + АЛУ

Главное понятие ИмпПП - **состояние** (хранится в памяти ЦП)

F: S->S - "такт" работы ЦПУ

S_0->S_1->S_2.....

"Главный" импЯП - ЯА

Императивная парадигма

Команды ЯА (точнее - машинные команды)

- пересылки между ОП и регистрами ЦПУ (MOV EAX, A)
- арифметико-логические команды (ADD EAX, ECX)
- команды управления, включающие в себя команды перехода (JMP F) и некоторые специальные команды (HALT)
- команды ввода/вывода (IN AH, PORT_NUM)

Вспомним учебник В.Н.Пильщикова по ЯА- объем глав говорит о (относительной) важности команд - главы 2,3,4

Сам ЯА содержит предложения 2 видов - директивы:

A DB ?

PORT_NUM EQU 80h

и команды (см. выше)

Императивная парадигма

Теперь посмотрим на ЛЮБОЙ императивный язык (например, Фортран или Паскаль)

- Оператор присваивания (V = 1 V :=1) - общий вид: V <= Expr
- Выражения, состоящие из операций, аргументы которых - операнды (константы, переменные) - A + B/C + 2.0 (на каком языке?)
- Операторы управления, включающие в себя циклы, ветвления, вызовы подпрограмм (+команды перехода) (Ф - 1 цикл, 1 ветвление и 4 оператора перехода)
- команды ввода/вывода - в современных языках ЯП их практически нет - ушли в станд. библиотеку. Паскаль - writeln(x,y); writeln; writeln(d:8:2) C - нет (см. stdio.h)

Предложения любого императивного языка делятся на 2 вида:

- объявления (типа, константы, переменной, подпрограммы.....)

REAL D

D:real

const N = 3

- операторы (см. выше)

Как видим - полная аналогия с ЯА, только более абстрактно и машинно-независимо

Императивная парадигма

Пара отступлений

Первое: замечание о терминологии

Оператор <=> statement (loop statement, assignment statement...)

Операция <=> operator (C++)

Почему? Долгая отечественная традиция (с 50-х гг) - операторные схемы Ляпунова и др. работы - перевод сообщения о языке Алгол-60 (Н.П.Трифонов и др.)

В чем отличие операторов и операций? В понятии ***побочного эффекта, то есть изменения состояния при вычислении***

Для операций - норма - отсутствие побочного эффекта,

для операторов - побочный эффект - смысл их существования

Императивная парадигма

Отступление 2: оператор присваивания в языке C

Императивная парадигма

Отступление 2: оператор присваивания в языке C

Его нет, так как существует более общая конструкция - оператор-выражение.

expression;

i++; f(); - имеет смысл, если выражение - с побочным эффектом

(то есть для C - ПЭ - норма, что само по себе не совсем нормально)

i; 5; f;   // бессмысленно, но вполне допустимо

i = 5   - выражение, так = в C - операция

i = 5; - аналог оператора присваивания

Откуда это все?

C - 1969 год

Самый "модный" язык в это время - Алгол (но 68)

Алгол-68 - последний из Алголов (фактически его вытеснил Паскаль)

Об ортогональности.

Одно из главных свойств Алгола-68 (второе главное свойство - формальное описание как синтаксиса, так и семантики вычислений - W-грамматики)

"Независимость языковых конструкций"

Допустимые конструкции имеют смысл в любом осмысленном контексте и могут быть взаимозаменяемыми

outer_context .... K....end_of_outer_context

V := Expr;

for i:=expr1 to expr2 step expr3 do statement

for i:=0 k:=n step (for j:=1 to N step 1 do S := S +a[j]) do P(i)

Операторы имеют вычисляемое значение, а выражения - побочные эффекты => они взаимозаменяемы

Императивная парадигма - реверс входной последовательности

Классическая задача обработки данных - последовательность шагов:

- Подготовить (ввести исходную последовательность)
- Обработать (собственно реверс)
- Завершить (вывести)

Причем 2 и 3 этапы можно и объединить, но только если реверсированная последовательность не нужна далее.

Императивная парадигма (язык C)

```
#include <stdio.h>
#define MAX_ELEMENTS 1024
char Input[MAX_ELEMENTS];
int main() {
        int current, count = 0;
        while ((current = getchar()) != EOF)
        if (count == MAX_ELEMENTS) {
                fprintf(stderr, "Слишком много символов");
                return 1;
        }  else
                Input[count++] = current;
        for (int i = count-1; i >= 0; i--)
                putchar(Input[i]);
        return 0;
}  // проблема - не соответствует условию задачи (где ограничения?)
```

Императивная парадигма (язык С)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_ELEMENTS 1024
char * pInput = NULL;
char Buffer[MAX_ELEMENTS];
int main() {
        int current, count = 0;
        pInput = (char*)calloc(1, sizeof (char));
        while (fgets(Buffer, sizeof Buffer, stdin)) {
                int oldCount = count;
                current = strlen(Buffer);
                count += current;
                pInput = (char*) realloc(pInput, count + 1);
                if (!pInput) {
                        fprintf(stderr, "Не хватает памяти");
                        return 1;
                }  else
                        strcat(pInput + oldCount, Buffer);
        }
        for (char * pBegin = pInput, * pEnd = pInput + count - 1;  pBegin < pEnd; ++pBegin, --pEnd) {
                char t = *pBegin; *pBegin = *pEnd; *pEnd = t;
        }
        puts(pInput); free(pInput);
        return 0;
}
```

Императивная парадигма (язык Go)

```
package main
import ("fmt"; "io/ioutil"; "os"; "strings")
func main() {
	rdr := os.Stdin
	b, err := ioutil.ReadAll(rdr)
	if err != nil {
		panic("bad input")
	}
	s := strings.Split(string(b),"")
	// если просто надо вывести последовательность
	for i := len(b) - 1; i >= 0; i-- {
		fmt.Print(s[i])
	}
}
```

Императивная парадигма (язык Go)

```
package main
import ("fmt"; "io/ioutil"; "os"; "strings")
func main() {
        rdr := os.Stdin
        b, err := ioutil.ReadAll(rdr)  // b - это не строка, а вырезка из байтов!!!
        if err != nil {
                panic("bad input")
        }
        s := strings.Split(string(b),"") // string(b) - вырезку в строку, далее делим строку на символы
        // для получения символа нельзя обращаться к строке по индексу (UTF-8)
        for i, j := 0, len(s)-1; i < j; i, j = i+1, j-1 {
        s[i], s[j] = s[j], s[i]
        }
        // вырезки из строк-символов объединим в строку и выведем
        fmt.Println(strings.Join(s,""))
}
```